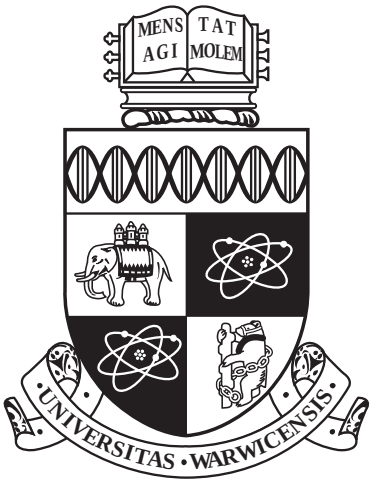


Mechanism design for fair allocation on uniform machines

by

Ruini Qu

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

Warwick Business School

September 2017

THE UNIVERSITY OF
WARWICK

Contents

Chapter 1

Introduction

1.1 Background and motivation

In traditional machine scheduling problems, a central decision maker, provided with all the relevant information about a system, is asked to derive an allocation scheme that optimizes some global objective, while simultaneously satisfying all the side constraints of the problem. However, since the emergence of the Internet as a computation platform, the assumption of information completeness does not hold anymore and algorithm designers are encouraged to reconsider the problem from a decentralized perspective. Most importantly, when decisions are made by independent agents, it is more likely that a rational agent will implement the strategy in such a way that maximizes their own interests, regardless of the overall system performance. Such situations require algorithm designers to not only focus on the global performance of the system, but also to take into account the strategic behaviour of the individuals involved.

Algorithmic Mechanism Design (AMD), a term coined by Nisan and Ronen (1999), specifically targets this kind of problem, where part of the input is under the control of selfish agents who do not have an incentive to tell the truth, unless truth-telling is for their own good. This type of design endeavours to merge the challenges from two classic disciplines: algorithm design in computer science, and mechanism design in game theory. The former emphasizes the importance of the computational efficiency of an algorithm, while ignoring the elements related to incentives; the latter, instead, normally yields game theoretic outcomes with poor computations. AMD, on the other hand, aims to present good game theoretic properties and good computational properties at the same time. Guided by the idea of AMD, research has

been conducted on scheduling problems with various models and various objectives. Among them, some of the most popular objectives include the minimization of the maximum completion time (also known as the makespan), and the maximization of the minimum completion time (also known as the cover). Minimizing the makespan is naturally related to efficiency, as it ensures that the entire job set is completed within the shortest possible time; maximizing cover, instead, embodies the concept of fairness from the machine owner's perspective in the sense that a machine will not get exemption due to its slowness. However, it can be argued that the fairness embodied by both objectives is only to a limited extent as they both can lead to extreme situations.

Fairness is an important social concept that has not been well considered in the literature of AMD. This is surprising given that "each person possesses an inviolability founded on justice that even the welfare of the society as a whole cannot override" (Rawls, 2009, pg.3). To concretely state the importance of fairness in the scheduling context, let us consider the problem faced by the U.S. Federal Aviation Administration. Billions of monetary losses are incurred as a result of unpredictable system delays each year. To improve this situation, proposals have been raised by scholars which have guaranteed more efficient schedules and claimed greater savings in costs. Unfortunately, few of those proposals have been implemented in practice, mainly because they fail to take the issue of fairness into consideration (Bertsimas and Patterson, 2000).

Fairness plays a key role in resource allocation, especially in socially oriented areas, including education, medical systems, and businesses. Although it is in the interest of the central authority to achieve system efficiency when allocating resources, individual players tend to care more about their own interests. If they cannot maximize their own benefits, then they at least want to be treated fairly. As a special case of resource allocation, machine scheduling problems also face similar challenges deriving from the players' desire for fairness. To achieve higher levels of fairness, we propose a new objective called minimizing the maximum deviation, which aims to minimize the maximum deviation between the completion time of each individual machine and the average completion time of the system, calculated as the sum of all the job sizes divided by the sum of all the machine speeds. To the best of our knowledge, this objective has not been considered by others before.

1.2 Literature review

1.2.1 Fairness

Considered to be cornerstones of a healthy society, the notions of equity, justice and fairness have been extensively studied from both a qualitative social analysis and a quantitative mathematical perspective. The fair treatment of individual social units is regarded as an important organizational goal within a wide variety of settings, including education, health care, business and government. However, it is not easy to quantify the notion of fairness in the absence of a proper standard and feasible framework. Also, due to the numerous existing interpretations of the term fairness, together with the differences in the nature of resource allocation problems, a universally agreed-upon definition of this notion is still lacking. Nevertheless, the literature presents a number of general theories of fairness which provide the basis to develop the majority of fairness schemes.

The oldest and probably one of the most famous theories of justice is represented by Aristotle's equity principle (Rowe and Broadie, 2002). Sometimes cited as the *desert theory*, Aristotle's rule of distributive justice states that goods should be apportioned in proportion to each claimant's contribution. Although this idea seems reasonable, critics of the theory argue that in order for the theory to be applicable, there must be a way to measure each claimant's contribution on a cardinal scale and that the resources must be arbitrarily divisible. Another prominent theory is developed from the classical utilitarianism, which assumes that an economic man balances his losses against his gains in order to fulfill his own interests. From the utilitarian view, a justice distribution is to achieve the maximum utilities of the sum, regardless how this sum of utilities is distributed among individuals. The principle has been widely examined in the area of welfare economics in the 19^{th} century. However, the clear absence of fairness and lack of a standard measurement for utility make it controversial. A third approach proposed by Rawls (2009) is based on a central principle, according to which the highest minimum utility level of each player needs to be guaranteed. In other words, this principle aims to ensure that the least well-off group in society becomes as well off as possible. A refined version of this principle is given by the lexicographic Rawlsian maximin rule. It follows the Rawlsian max-min fairness, but if the worst off players in two distributions are equally well off, the principles compare the utilities of the second worse off person in each distribution, and so on, until a difference shows, then the principles select the one that maximizes the utility of that player. Finally, Nash developed the Nash

standard of comparison, which measures the percentage change of the claimants' utilities when a small amount of resources is transferred between two claimants. Such a transfer is only justified when the gainer's utility increases by a larger proportion than the decrease in the losers' utility. For more detailed information on these theories, we refer readers to Young (1995) and Sen (1973).

We define *deviation* as the absolute difference between the completion time of a machine and the average completion time of the system. The aim to minimize the maximum deviation among the machines embodies the concept of max-min fairness, as well as its dual definition, min-max fairness. Developed from Rawlsian justice, max-min fairness is among the oldest and most widely applied fairness criteria. Initially discussed in the domain of communication networks (Megiddo, 1974), max-min fairness saw its application in window flow control protocols (Hahne, 1991), before becoming very popular in both wired and wireless networks (Charny, 1994; Ma and Steenkiste, 1997; Roborts, 1994; Nandapogal et al., 2000; Sarkar and Tassiulas, 2000; Huang and Bensaou, 2001; Sridharan and Krishnamachari, 2004). More specifically, max-min fairness was first studied in single-source fractional flow networks. However, later Kleinberg et al. (1999) raised the idea of unsplittable flows and showed that when restricted by unsplittable flows, the original problem becomes NP-complete. Their work highlighted the computational difficulty of deciding a max-min fair allocation. A direct result of their conclusions is a shift in research focus from exact algorithms to approximation algorithms with strong guarantees (Kumar and Kleinberg, 2000; Afek et al., 1996; Goel et al., 2000).

Definitions of max-min fairness have been described in slightly different ways in different problem settings. Nevertheless, the logic remains unchanged. An allocation is *max-min fair* if an increase in the utility of one individual can only be achieved by reducing the utility of other individuals whose utility is already smaller. In a max-min fair schedule, no one is able to further increase its utility because the objective is in favour of the worse off individuals. Among all works in this field, studies of max-min fair allocation of indivisible goods are the most related to this thesis. Also known as the Santa Claus problem, the max-min fair allocation problem was considered in the scheduling context by Bezáková and Dani (2005), whose rounding Assignment-LP was later improved by a stronger linear programming relaxation called Configuration-LP (Bansal and Sviridenko, 2006). An alternative rounding approach was initiated by Asadpour et al. (2008). Focusing on the restricted version of the problem, they prove that the integrality gap of the Configuration-LP is

at most 4 from a hypergraph matchings perspective. Inspired by their local search procedure, the most recent work from Annamalai et al. (2017) proposes a purely combinatorial algorithm which further improves the approximation guarantee.

The main problem of the max-min fair criterion is that its corresponding allocation is not necessarily *Pareto optimal*. In other words, it is possible to increase the utility of one individual without decreasing that of others in a max-min fair allocation (Massoulié, 2007). However, this can be solved by introducing the notion of leximin ordering, the formal definition of which will be introduced in the next chapter. It has been proved that the leximin social welfare optimum always yields Pareto optimality (Mas-Colell et al., 1995). The biggest criticism about max-min fairness and its lexicographical version is that it only concentrates on the worst off individual (or the kth worst off under the lexicographical max-min) while ignoring the rest.

By comparison, min-max fairness receives much less attention in network studies. It has been shown that in general there is a fairness gap between the performance achieved under min-max fairness and under max-min fairness (Boche et al., 2007). Earlier work conducted by Deuermeyer et al. (1982) also suggests that the study of a scheduling problem under the min-max objective function is not simply determined by the dual form of the max-min function. Boche et al. (2007) also characterize a subclass of networks in which max-min fairness and min-max fairness can be achieved simultaneously.

A natural concern over the issue of fairness is whether or not the achievement of fairness is incompatible with global utility optimality. The conjecture that a fairer allocation is also less efficient has been prompted by massive examples in the resource allocation literature, ranging from wired networks (Mo and Walrand, 2000; Bonald and Massoulié, 2001) and wireless networks (Luo et al., 2004; Srinivasan and Somani, 2003) to economics (Bulter and Williams, 2002). These studies seem to conclude that fairness equals inefficiency. The work by Tang et al. (2004), nevertheless, suggests the opposite. By investigating the exact characteristics of the trade-off between fairness and throughput in general networks, they produce a set of counter examples where a fairer allocation is revealed to be always more efficient. Their work has not only changed the stereotype of the inherent conflict between fairness and efficiency, but it has also encouraged efforts towards generating algorithms that guarantee the coexistence of the two seemingly conflicting objectives.

The trade-off between fairness and efficiency has been widely discussed in air traffic management (Bertsimas and Patterson, 2000; Ball et al., 2007; Ball and Lulli, 2004; Barnhart et al., 2012), medical settings (Williams, 1985; Su and Zenios, 2006; Bisias et al., 2012), and call center designs (de Véricourt and Zhou, 2005; Luh and Viniotis, 2002; Bonald et al, 2006). However, this trade-off is only discussed from a soft qualitative perspective. It is not until the introduction of the concept of *price of fairness* (Bertsimas et al., 2011) that the trade-off between fairness and efficiency can be truly quantified.

More specifically, Bertsimas et al. (2011) describe a canonical resource allocation problem as a problem involving n players and a central decision maker, who assigns scarce resources among the players. Each player defines a utility function based on their own preference, and the final utility they derive depends on this utility function, as well as on the allocation chosen by the central decision maker. By using the classical utilitarian principle, Bertsimas et al. (2011) define a fully efficient allocation as an allocation that maximizes the sum of the utilities of all players. The price of fairness can then be defined as the relative performance loss under a 'fair' allocation compared to the above fully efficient allocation.

1.2.2 Mechanism design

Retrospectively, the problem of resource allocation has been discussed under complete information settings, where players behave in line with the instructions received from a central authority (Ramakrishnan et al., 1987; Afek et al., 1996). With the emergence of diverse and decentralized computing environments, researchers have gradually realized that it is not always reasonable to assume a universal implementation of any given algorithm. In a decentralized setting, decisions and actions are taken by selfish agents with conflicting preferences that need to be aggregated to form one socially desirable outcome. Such mechanisms for preference aggregation include, but are not limited to, auctions, divorce settlement procedures, voting protocols, and collaborative ratings.

Lack of control from a central authority creates the opportunity for individual agents to *manipulate* the system by misreporting their preferences if, by doing so, they can mislead the mechanism to choose an outcome that is more favourable to the agent than the outcome that would have been selected otherwise. Manipulability is a pervasive problem in mechanism design which is undesirable given that insincere in-

formation can result in an undesirable overall social outcome. To solve the problem of manipulability, early efforts were focused on social choice functions. Unfortunately, with regard to this problem, negative results were successively achieved both by Gibbard (1973) and Satterthwaite (1975) independently. Their theorem states that under any nondictatorial preference aggregation scheme, if there are at least three possible outcomes, then an agent is better off by reporting strategically under some preferences. Consequently, the theorem rules out all hope of designing a truthfully implementable social choice function without any external incentives.

Mechanism design aims at avoiding this negative result by introducing various modifications of the model. One of the successful modifications to this end is given by monetary compensation. Nevertheless, when in some social settings money transfer is deemed to be undesirable, an alternative approach with important breakthroughs is determined by modifying the assumption of unrestricted preference domains. The most famous positive result of this type is represented by the Vickrey-Clarke-Groves (VCG) mechanisms, which aim at optimizing (weighted) social welfare. It has been shown that if the valuation space has full dimensionality, or unrestricted domains, the VCG mechanism is the only class that can be truthfully implemented. In addition, as long as the target function maximizes the sum of all the agents' utilities, VCG presents no restrictions on the valuation functions of the agents.

Based on the main concern of the design, traditional mechanism designs fall into two different strands. One is represented by optimality, which maximizes the expected revenue of the seller. The other is given by efficiency, which pursues social efficiency instead of revenue maximization. In the following sections, a well-known application and the corresponding results for each type are briefly presented.

Auctions

A seller can serve exactly one of the many bidders, each of whom has a private valuation for the same item. The seller's aim is to maximize the gain from selling the item. The most famous result in auction theory is given by Vickrey's second price auction (1961). The mechanism states that whoever offers the highest valuation for the request pays the second highest price offered. Applications of mechanism designs to auction theory are among the richest (Myerson, 1981; Che and Gale, 1996). We refer the readers to Klemperer (1999) for a comprehensive review on early studies. The emergence of cloud computing and electronic markets has triggered a renewed surge of studies in the field (Lin et al., 2010; Zhang et al., 2016).

Public Projects

The problem of public projects relates to how much each user, who commonly benefits from the upgrade of some public facilities, is charged if such a decision is made. The solution to this problem is known as the Clarke tax (1971), which states that once an upgrade decision is made, each user pays the portion of their bid that makes a difference to the outcome. This solves the so-called *free rider* problem which was considered to be unsolvable prior to the introduction of Clarke tax.

The mechanism considered in this thesis falls under the umbrella of the *direct revelation* model, where the only action an agent needs to take is to report its type (Gui et al., 2004; Lavi et al., 2003; Saks and Yu, 2005). Among all the models falling into this type, machine scheduling was popularized thanks to the work by Nisan and Ronen (1999).

Just like in the scheduling models under complete information, the makespan minimization function is the objective which has been the most commonly studied in the field of AMD. Unfortunately, this objective does not take the classical utilitarian form; as a result, VCG cannot be applied. Nisan and Ronen (1999) focus on the truthful mechanisms in machine scheduling with unrelated machine agents. They conclude that there is no truthful mechanism that minimizes the makespan in this setting. Archer and Tardos (2001) examine a similar model for related machine scheduling. This time, however, a necessary and sufficient condition is derived for an allocation algorithm to be truthfully implementable. The condition, which we will introduce in detail in the following chapter, acts as a theoretical basis for the majority of, if not all, the design mechanisms of this type, including the studies in this thesis. Furthermore, Archer and Tardos (2001) show that the optimal fractional solution qualifies the truthful condition, but only in expectation, if the partial jobs are allocated randomly between the two machines they assigned to. This is the first positive result in AMD studies, even though the algorithm embodies a weaker notion of truthfulness, in the sense that agents maximize their expected, rather than their actual utilities. By rounding the speed of the machines, Andelman et al. (2005) provide the first deterministic truthful mechanism, which achieves 5-approximation. Using a similar rounding technique, Kovács (2005) improves the approximation ratio to 3 on the basis of the well-known greedy algorithm – Longest Processing Time (LPT). Auletta et al. (2004) provide an alternative approach to combine optimality and truthfulness. The approach assigns jobs in two batches: the first batch contains

a limited number of big jobs which are assigned by an exact truthful algorithm; the second batch contains the rest of the jobs, which are assigned by a greedy truthful algorithm. Their approach yields a family of deterministic polynomial-time truthful $(4+\varepsilon)$-approximation mechanisms for any fixed number of machines. The problem of minimizing the maximum completion time is solved by proving the existence of the truthful polynomial time approximation scheme (PTAS) in both the randomized and the deterministic settings, as in Dhangwatnotai et al. (2011) and Christodoulou and Kovács (2013), respectively.

Following a natural shift from the makespan minimization problem, researchers started to target other objective functions, and unsurprisingly, cover maximization was the one which attracted their attention. The cover maximization problem was first touched upon by Epstein and van Stee (2010). They propose a new technique which reduces the number of jobs while remaining close to the optimal solution at the same time. The monotone PTAS of their work is linear in the number of jobs. Christodoulou et al. (2010) provide an approximation guarantee of $2+\varepsilon$, which is a significant improvement from Epstein and van Stee's upper bound of $\min\{m, (2+\varepsilon)s_m/s_1\}$. Finally, Epstein et al. (2013) claim that a wide class of scheduling problems (including makespan minimization, cover maximization, and minimizing the ℓ_p-norm of the machine workloads vector) can be addressed by means of a unified framework that outputs deterministic monotone PTAS for uniformly related machines.

It is noticeable that in a strategic scheduling model, apart from machine agents, we can also assume to have job agents, as in the model described by Koutsoupias and Papadimitriou (1999). One fundamental assumption of such research is that part of the job set is unknown and will only be revealed over time. Porter (2004) studies the truthful mechanism for online scheduling with job agents on a single machine. Heydenreich et al. (2010) look at the problem with m parallel machines. Due to the focus of our research, however, we will not investigate this branch of studies deeply.

1.2.3 Machine scheduling

Scheduling resources and tasks to processors is a decision that is made regularly in both the manufacturing and the service industries. For example, in the context of airports, each arriving and departing plane needs to be assigned to one of the gates; while in schools, teaching facilities such as classrooms and projectors need to be

assigned at the beginning of each term according to the curricula. In this thesis, *job* is used to represent any form of resource or task await for allocation, while *machine* refers to a processor in general.

Both jobs and machines can present different characteristics, and the combination of these different characteristics leads to different machine scheduling models. For example, machines might be identical or operate at different speeds; the processing of jobs might require single or multiple resources; or there might be precedence constraints between jobs. We refer readers to Chen et al. (1998) for a nicely presented review on machine scheduling. To the best of our knowledge, the application of game theory is only limited to a few machine scheduling models. Moreover, a truthful mechanism does not exist in the strategic version of makespan minimization for scheduling with unrelated machines (Nisan and Ronen, 1999). This thesis focuses on the assignment of off-line jobs to uniform parallel machines without preemption. More specifically, machines are considered as identical except that each machine processes jobs at a different speed. All the jobs and their associated features are fully released before an allocation decision is made. Once a machine starts to process a job, it cannot process another job unless it finishes first the job at hand. In other words, a schedule is only feasible if each job is assigned to one machine, and each machine processes one job at a time without any interruptions.

1.3 Contributions and thesis structure

This work emphasizes the importance of fairness in machine scheduling problems by proposing a new objective function. The goal of the deviation minimization objective is to achieve more evenly distributed workloads among machines by ensuring that each machine's completion time is not too far away from the average completion time. The model is formally presented in the following chapter, together with the basic concepts, notations, and fundamental theories.

In Chapter 3, we examine the performance of the existing approximation algorithms. We found that a truthful α-approximation algorithm for the makespan minimization problem can be directly applied to the new objective. The algorithm will remain α-approximate for the deviation minimization problem if $\alpha \geq 2$, and will become 2-approximate if $\alpha < 2$. The second part of the chapter focuses on LPT* – a fast 3-approximation algorithm proposed by Kovács (2005). A tighter bound for LPT* of 2.8 is found.

Chapter 4 considers the exact algorithm for the new objective. We show that the problem cannot be truthfully implemented with any exact algorithm. As a result, the existing approaches for generating a truthful PTAS are not applicable to the deviation minimization problem. We propose an alternative tie breaking rule which successfully narrows down the type of instances where truthfulness is not guaranteed. Computational results imply that if the central authority requires the agents to report their speeds in a rounding format, our proposed algorithm becomes truthful.

Chapter 5 takes an alternative view and discusses the issue of fairness among jobs with due dates. The traditional maximum tardiness minimization objective is employed for this purpose. Due to the lack of connection between job sizes and job due dates, the tardiness objective cannot be truthfully implemented with an exact algorithm either. From the observation that the only difference between the longest processing time and the earliest due date is the job sequence, we propose the two-stage LPT*, which remains the truthful feature of LPT*, and produces a much improved performance in minimizing the maximum tardiness.

Finally, the final remarks and the conclusion of the thesis are presented in Chapter 6.

Chapter 2

Preliminaries

In this chapter we formally present our model and the objective function. We define the basic notation used throughout the thesis under both the scheduling and the game theoretic contexts respectively, as well as the mathematical form of the deviation minimization objective. A tailored approximation ratio to measure algorithm performance is introduced followed by examples that compare the new objective function with the existing ones.

2.1 Basic concepts and notations

2.1.1 Scheduling model

We consider machine scheduling problems with the following characteristics. There is a set of jobs $J = \{J_1, \ldots, J_n\}$, and each job must be assigned to one and only one machine from a set of machines $M = \{M_1, \ldots, M_m\}$ with no preemption. Job J_j is associated with a size of $p_j > 0$, machine M_i is characterized by speed $s_i > 0$, and the processing time of J_j on M_i equals p_j/s_i. For simplicity, we sometimes refer J_j to its job length p_j. An allocation of jobs to machines is to partition the jobs into m subsets that are mutually exclusive and collectively exhaustive. In a feasible allocation, denoted by π, each job must be completely processed without interruption and no machine can process more than one job at a time. Π denotes the entire feasible allocations from J to M.

An algorithm $A : J \to M$ is defined as a set of rules to be followed in allocating the entire job set onto machines. Implementing A to an instance I, the sum of all the processing times being assigned to M_i is called the workload of M_i, denoted by $W_i^A(I)$. The completion time of M_i is computed as $T_i^A(I) = \frac{W_i^A(I)}{s_i}$. Notation I

is often omitted for simplicity if doing so causes no confusion. We define cover as the smallest completion time of instance I under A, denoted by $T^A_{\min}(I) = \min_i\{T^A_i(I)\}$, and makespan as the largest completion time, denoted by $T^A_{\max}(I) = \max_i\{T^A_i(I)\}$.

We define a *fair schedule* as an allocation in which each machine finishes its assigned workload by the same time length T_0, where T_0 is known as the average completion time and is computed as

$$T_0 = \frac{\sum_{j\in J} p_j}{\sum_{i\in M} s_i}.$$

It is important to note that such a fair schedule is not always feasible because the jobs to be assigned are unsplittable.

To generate a feasible schedule that is close enough to the fair schedule, we introduce the concept of *deviation*. The deviation of M_i under allocation π is defined as

$$\mathrm{dev}_i(\pi) = |T_i(\pi) - T_0|,$$

and denote by

$$\mathrm{dev}(\pi) = \max_i\{\mathrm{dev}_i(\pi)\}$$

the maximum deviation of π. Likewise, $\mathrm{dev}^A(I)$ denotes the maximum deviation from implementing algorithm A to instance I. The deviation minimization objective function is therefore,

$$\min_{\pi\in\Pi} \max_i\{\mathrm{dev}_i(\pi)\}.$$

Finally, we provide the definitions for the bottleneck machine and lexicographical comparison respectively as follows.

Definition 2.1 (bottleneck machine)**.** *A machine is a bottleneck machine of a schedule if it achieves the optimal value of an objective function.*

Definition 2.2 (lexicographical comparison)**.** *A vector $(v_1, \ldots, v_m)$ is lexicographically smaller (bigger) than $(\bar{v_1}, \ldots, \bar{v_m})$ if, for some l, $v_l < \bar{v_l}$ $(v_l > \bar{v_l})$ and $v_i = \bar{v_i}$ for all $i < l$.*

2.1.2 Mechanism design model

In the strategic version of machine scheduling, one crucial element is given by the interactions between a set of independent agents, which can represent either the

set of jobs or the set of machines. This thesis focuses on the machine-agent model, where each $M_i \in M$ is controlled by an independent agent i, who considers machine speed s_i as its private information, also known as the agent's *type*. In the game, the only action that an agent needs to carry out is to report its speed. Therefore, the agent's *action* is denoted by its reporting speed b_i. Beside the private information, usually there is also *public information* that is shared across the system, such as the number of machines or the size of the jobs.

The scheduling decision made by the central authority relies on both the public information and the actions taken by each agent. Given that the omission of public information hardly causes problem of ambiguity, it has been excluded from our model. The output algorithm, denoted by $A(b) \in \Pi$, computes a schedule of jobs to machines based on the agents' actions, which in turn are influenced by the agents' types. We define *strategy* x_i as a mapping from agent i's type space into its action space, i.e. $x_i : s_i \to b_i$. It is important to note that i chooses its strategy without being told what are the types of the other agents. Let b_{-i} denote the vector of the reporting speeds of all the other agents, except for i. The entire vector of bids b can be written as (b_i, b_{-i}), which is equivalent to $(x_i(s_i), b_{-i})$.

Different outcomes are valued differently by an agent. We express this by the *valuation* of an agent for a certain schedule, which normally depends on the true value of the agent's type. Therefore, we denote agent i's valuation for outcome π as $v_i(\pi|s_i)$. We assume that an agent always prefers an outcome with a higher value. Concretely, if schedule π is preferred over π' by agent i, then $v_i(\pi|s_i) > v_i(\pi'|s_i)$. Under the scheduling context, it is assumed that a machine incurs a *cost* in processing the workload to which it has been allocated. It is further assumed that the cost for agent i to process one unit of job length equals the inverse of its actual speed, i.e., $\text{cost}_i = \frac{1}{s_i}$. Complying with the assumptions, agents view their workloads as a burden; hence they would prefer allocations that assign them with fewer workloads. Accordingly, agent i's valuation towards schedule π is the total costs in processing its workload under π, i.e., $v_i(\pi|s_i) = -\frac{W_i(\pi)}{s_i}$, which takes negative values.

To improve the overall quality of a schedule, it is common to manipulate agents' strategies by introducing monetary *payments*. In machine scheduling context, the payments are made by the central authority to the agents based on their reporting speeds in order to induce a desired behaviour. Let $\xi_i(b_i, b_{-i}) \in \mathbb{R}$ denote the payment function of agent i. Therefore, the *mechanism* model, denoted by μ, consists

of an allocation algorithm A as well as a payment scheme ξ, i.e., $\mu = (A, \xi)$. With no further information on b_{-i}, a strategic agent will act in such a way that the schedule generated by the output function always maximizes its overall utility no matter what action is taken by the other agents. Such strategy is known as the *dominant strategy*, which presents the following formal definition.

Definition 2.3 (dominant strategy equilibrium)**.** *A strategy vector $x \in X$ represents a dominant strategy equilibrium, if for all agents i, for all types s_i of agent i, for all actions b_{-i} of the other agents, and all alternative actions b_i of agent i, the following holds:*

$$v_i(A(x_i(s_i), b_{-i})|s_i) - \xi_i(x_i(s_i), b_{-i}) \geq v_i(A(b_i, b_{-i})|s_i) - \xi_i(b_i, b_{-i}).$$

Based on the dominant strategy equilibrium, the truthful mechanism considered in this thesis is defined as follows.

Definition 2.4 (truthful and truthfully implementable)**.** *A direct revelation mechanism is truthful if the strategy vector x in which each agent truthfully reports its type is a dominant strategy equilibrium, i.e., $x_i(s_i) = s_i$. An allocation algorithm A is said to be truthfully implementable if we can find a payment rule ξ such that the mechanism $\mu = (A, \xi)$ is truthful.*

Now let us introduce a necessary and sufficient condition for a truthful mechanism in the context of related machine scheduling with machine agents.

Theorem 2.1 (Archer and Tardos, 2001)**.** *An allocation algorithm is truthfully implementable if and only if for all agents i and all $b_{-i} \in B^{m-1}$, where B^{m-1} denote the action space of all agents except i, the workload function $W_i(b_i, b_{-i})$ is an increasing function of b_i. If this is the case, then the following payments yield a truthful mechanism:*

$$\xi_i(b_i, b_{-i}) = h_i(b_{-i}) + \frac{1}{b_i} W_i(b_i, b_{-i}) - \int_0^{b_i} W_i(u, b_{-i}) du.$$

Here, h_i denote arbitrary functions that are independent of b_i.

The theorem says that an allocation algorithm is truthfully implementable if and only if the workload of an agent does not decrease as its reporting speed increases. In addition, the theorem specifies the form for the payment functions in a truthful mechanism. This theorem is the most famous in AMD and serves as the foundation for many subsequent mechanism designs for related parallel machine scheduling,

since it neatly separates the problem of designing the allocation algorithm from the payment scheme.

2.1.3 Approximation algorithms

The problem of scheduling n independent jobs on m uniform machines is known to be NP-complete (Lenstra et al., 1977). For such problems, it is generally considered to be unlikely for a polynomial time bounded algorithm to exist. Therefore, a common practice is to solve an NP-complete or harder problem by means of efficient heuristic methods in the hope of finding "good" solutions rather than attempting to find optimal ones. This study focuses on designing approximation algorithms with provable guarantees on performance.

Definition 2.5 (r-approximation)**.** *Let K denote an optimization problem. Let A be an algorithm such that for any instance I of K, A computes a feasible solution with cost $cost^A(I)$. A represents an r-approximation for K if for any instance I of K*

$$\frac{1}{r} \leq \frac{cost^A(I)}{cost^*(I)} \leq r,$$

where $cost^*(I)$ denotes the optimal value and $r \geq 1$.

As for the deviation minimization problem, the objective outputs the maximum deviation of each machine's completion time from the average completion time. According to a direct application of the r-approximation definition, the performance of algorithm A for the deviation minimization problem is measured by

$$\rho^A(I) = \frac{\mathrm{dev}^A(I)}{\mathrm{dev}^*(I)};\ \rho^A = \sup_I \rho^A(I).$$

One problem with the above definition is that, in some instances, jobs can be evenly distributed among machines so that the fair schedule is realized. If this is the case, then the optimal deviation $\mathrm{dev}^*(I)$ equals zero, and ρ^A becomes infinity. To overcome this weakness, we add the average completion time to both the numerator and the denominator. Therefore, the approximation ratio of A over the deviation minimization problem is adjusted to

$$\rho^A(I) = \frac{\mathrm{dev}^A(I) + T_0(I)}{\mathrm{dev}^*(I) + T_0(I)};\ \rho^A = \sup_I \rho^A(I).$$

2.2 Comparing the objective functions

This section clarifies the significance of the new objective function. As mentioned in the literature review, fairness is a fundamental issue in many resource allocation problems. When we talk about fair allocations in machine scheduling, more often than not we refer to the allocation that maximizes the minimum completion time as studied by Epstein and van Stee (2010). However, we argue that the fairness achieved under the cover maximization objective is limited. We illustrate this argument with a simple but extreme example.

Example 2.1. *Consider an instance of two machines, with speeds* $s_1 = 1$ *and* $s_2 = 10$*. Assign two jobs with the same size of* 5 *under the three different objectives respectively and display the corresponding workloads in the table below.*

	Machine 1	Machine 2
Cover maximization	5	5
Makespan minimization	0	10
Deviation minimization	0	10

Table 2.1: **Workloads under different objective functions** ($p_1 = p_2 = 5$)

In the above example, the schedule that maximizes the cover discriminates the slow machine in the sense that it takes the slow machine 10 times longer to complete its allocated workload than the fast machine. As a result, we argue that the other two objectives, i.e., the makespan minimization and deviation minimization, yield a fairer solution. Now let us consider another example.

Example 2.2. *Consider an instance with the same machines as above. This time assign two jobs of sizes* 1 *and* 8.9 *respectively. The workloads of each machine are summarized below.*

	Machine 1	Machine 2
Makespan minimization	0	9.9
Cover maximization	1	8.9
Deviation minimization	1	8.9

Table 2.2: **Workloads under different objective functions** ($p_1 = 1, p_2 = 8.9$)

As shown in the table, in order to decrease the makespan from 1 to 0.99, the slow machine is left to be slack under the makespan minimization objective, even though assigning the unit job to it is intuitively a fairer solution.

The last example shows that although the deviation minimization function is a simple combination of the makespan minimization and cover maximization functions, the schedule it generates is different from the old objectives.

Example 2.3. *Consider an instance with four machines, with speed* $s_1 = 27$, $s_2 = 20$, $s_3 = 17$, *and* $s_4 = 2$. *Assign a total number of* 7 *jobs to these machines under three objectives, i.e., the makespan minimization, the cover maximization and the deviation minimization, respectively. Table 2.3 presents the sizes of each job and their allocations under different objective functions.*

The completion time of each machine is presented in the last row of Table 2.3. Under the deviation minimization function, there is no significant difference in the machines' completion times, which implies that all the machines spend a similar amount of time to complete their assigned workload. However, if jobs are allocated under the makespan minimization objective, the other three machines will envy M_4, who receives zero workload. Alternatively, if the allocation follows the cover maximization function, M_4 who spends double the time to complete its work, will envy the other three machines instead. Inarguably, the allocation under the deviation minimization objective causes the lowest level of envy and therefore is the fairest among the three.

		Makespan				Cover				Deviation			
		M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
Jobs	$p_1 = 50$		√				√			√			
	$p_2 = 35$	√				√				√			
	$p_3 = 34$	√						√			√		
	$p_4 = 27$			√		√						√	
	$p_5 = 10$			√					√			√	
	$p_6 = 9$			√				√			√		
	$p_7 = 7$	√				√							√
Workload		76	50	46	0	69	50	43	10	85	43	37	7
Completion time		2.82	2.5	2.71	0	2.56	2.5	2.53	5	3.15	2.15	2.18	3.5

Table 2.3: **Job allocation under different objectives**

Chapter 3

Approximation algorithms

In this chapter, we examine the performance of some selected existing greedy algorithms on the deviation minimization problem from two perspectives: (a) the distance of the returned solution from the optimal fair schedule and (b) truthfulness in soliciting private information on speeds. Based on the initial observations, a general rule is derived which describes the relation between the deviation minimization objective and the makespan minimization objective. During this process, we reduce the bound of an existing algorithm from 3 to 2.8.

3.1 Longest processing time

Due to its advantage in computational efficiency, Longest Processing Time (LPT) has gained its popularity in machine scheduling studies. The greedy algorithm gives decent performance for both makespan minimization and cover maximization problems. Considering the deviation function is a combination of the two, it is a natural process to examine the performance of LPT on our new objective function.

The algorithm first orders the jobs according to non-increasing job sizes, and every time it assigns the next job in the ordered list to the machine which finishes earliest with the jobs assigned to it so far. According to Graham (1969) and Deuermeyer et al. (1982) respectively, for any instance I with identical machines, we have

$$\frac{T_{\max}^{\mathrm{LPT}}(I)}{T_{\max}^{*}(I)} \leq \frac{4}{3} - \frac{1}{3m}; \tag{3.1}$$

$$\frac{T_{\min}^{*}(I)}{T_{\min}^{\mathrm{LPT}}(I)} \leq \frac{4m-2}{3m-1}. \tag{3.2}$$

For uniform machines, according to Gonzalez et al. (1977), we have

$$\frac{T_{\max}^{\mathrm{LPT}}(I)}{T_{\max}^{*}(I)} \quad < \quad 2.$$

3.1.1 Bound LPT for identical machines

As a touchstone for the problem, we first consider a simplified version, where all machines have the same speed. We show that LPT maintains its performance in this specific case for the deviation function. Notice that when $m = 1$, the problem becomes trivial as all jobs have to be allocated to that single machine, resulting in the same deviation value for any algorithm. Therefore, we only consider $m \geq 2$ in all our proofs.

Proposition 3.1. *For scheduling m identical machines, we have $\rho^{LPT} \leq \frac{4}{3} - \frac{1}{3m}$ for the deviation minimization problem.*

Proof. Assume to the contrary that there is an instance I such that

$$\rho^{\mathrm{LPT}}(I) > \frac{4}{3} - \frac{1}{3m}. \tag{3.3}$$

By normalizing the job lengths we can assume without loss of generality that

$$\mathrm{dev}^{*}(I) + T_0(I) = 3m, \tag{3.4}$$

$$\mathrm{dev}^{\mathrm{LPT}}(I) + T_0(I) > 4m - 1. \tag{3.5}$$

With ρ^{LPT} understood as $\rho^{\mathrm{LPT}}(I)$, we omit I from our notation for simplicity in the remainder of our proof. According to the definition, we have

$$T_{\max}^{*} \leq \mathrm{dev}^{*} + T_0 = 3m. \tag{3.6}$$

If $\mathrm{dev}^{\mathrm{LPT}} = T_{\max}^{\mathrm{LPT}} - T_0$, then (3.6) implies

$$\rho^{\mathrm{LPT}} = \frac{\mathrm{dev}^{\mathrm{LPT}} + T_0}{\mathrm{dev}^{*} + T_0} \leq \frac{T_{\max}^{\mathrm{LPT}}}{T_{\max}^{*}},$$

which together with (3.3) contradicts inequality (3.1). Hence we must have

$$\mathrm{dev}^{\mathrm{LPT}} = T_0 - T_{\min}^{\mathrm{LPT}}, \tag{3.7}$$

which together with (3.5) implies $2T_0 - T_{\min}^{\mathrm{LPT}} > 4m - 1$, or

$$T_0 > 2m - \frac{1}{2} + \frac{T_{\min}^{\mathrm{LPT}}}{2}. \tag{3.8}$$

According to (3.4), $T_0 \leq 3m$, which together with (3.8) implies

$$T_{\min}^{\mathrm{LPT}} < 2m + 1. \tag{3.9}$$

Now we wish to derive an equation similar to (3.7) with LPT replaced by OPT. Noticing that $T_{\max}^* \leq 3m$ from (3.6), with (3.8) we have

$$T_{\max}^* - T_0 \leq 3m - T_0 < m + \frac{1}{2} - \frac{1}{2} \cdot T_{\min}^{\mathrm{LPT}}. \tag{3.10}$$

On the other hand, combining (3.2) and (3.8) leads to

$$\begin{aligned} T_0 - T_{\min}^* &> 2m - \frac{1}{2} + \frac{T_{\min}^{\mathrm{LPT}}}{2} - \frac{4m-2}{3m-1} \cdot T_{\min}^{\mathrm{LPT}} \\ &= 2m - \frac{1}{2} - \frac{5m-3}{6m-2} \cdot T_{\min}^{\mathrm{LPT}}. \end{aligned} \tag{3.11}$$

Combining (3.10) and (3.11) gives

$$\begin{aligned} &(T_0 - T_{\min}^*) - (T_{\max}^* - T_0) \\ &\quad > \left(2m - \frac{1}{2} - \frac{5m-3}{6m-2} \cdot T_{\min}^{\mathrm{LPT}}\right) - \left(m + \frac{1}{2} - \frac{1}{2} \cdot T_{\min}^{\mathrm{LPT}}\right) \\ &\quad = m - 1 - \frac{m-1}{3m-1} \cdot T_{\min}^{\mathrm{LPT}} \\ &\quad > m - 1 - \frac{m-1}{3m-1}(2m+1) = \frac{(m-1)(m-2)}{3m-1} \geq 0, \end{aligned}$$

where the second inequality is due to (3.9). Therefore, according to the definition we have $\mathrm{dev}^* = T_0 - T_{\min}^*$ as desired. Now this equation together with its counterpart (3.7) implies that

$$\rho^{\mathrm{LPT}} = \frac{\mathrm{dev}^{\mathrm{LPT}} + T_0}{\mathrm{dev}^* + T_0} = \frac{2T_0 - T_{\min}^{\mathrm{LPT}}}{2T_0 - T_{\min}^*},$$

which together with (3.3) implies

$$(4m-1)T_{\min}^* > 3mT_{\min}^{\mathrm{LPT}} + 2(m-1)T_0,$$

in which we replace $T_{\min}^*$ and T_0 with their bounds in (3.2) and (3.8) and obtain $T_{\min}^{\mathrm{LPT}} > 3m - 1$, which is in direct contradiction with (3.9), with which we complete

our proof of the proposition. □

The following example shows that the bound we derived in Proposition 3.1 is tight.

Example 3.1. *Consider an instance with m identical machines, and $n = 2m + 1$ jobs with processing times defined as*

$$p_j = \begin{cases} 2m - \lfloor (j+1)/2 \rfloor & , \quad j = 1, 2, \ldots, 2m \\ m & , \quad j = 2m+1 \end{cases}$$

Notice that in this way, jobs are automatically sorted in non-increasing order to their sizes, thus simplify the implementation of LPT. The average completion time

$$T_0 = \frac{\sum_{j=1}^{2m} [2m - \lfloor (j+1)/2 \rfloor] + m}{m} = 3m.$$

Machines	M_1	M_2	M_3	M_4	$\cdots$	M_m
Jobs	$2m-1$	$2m-1$	$2m-2$	$2m-2$	$\cdots$	$2m - \lfloor \frac{m+1}{2} \rfloor$
	m	m	$m+1$	$m+1$	$\cdots$	$2m - \lfloor \frac{m+2}{2} \rfloor$
	m					

Table 3.1: **Allocation of jobs to machines in LPT**

As shown in Table 3.1, LPT assigns the first m biggest jobs to machines in increasing order of machine indices, and the next m biggest jobs in reverse order of machine indices, followed by the final job being assigned to M_1. Clearly, each machine receives a workload of $3m-1$, except for M_1 whose workload is $4m-1$. Thus, the maximum deviation

$$dev^{LPT} = \max\{(4m-1) - 3m, 3m - (3m-1)\} = m - 1.$$

Table 3.2 presents the optimal allocation of jobs to machines for this instance. As can be seen, jobs can be evenly allocated so that each machine has the same processing time, which indicates $dev^* = 0$. Accordingly,

$$\rho^{LPT} = \frac{dev^{LPT} + T_0}{dev^* + T_0} = \frac{4m-1}{3m}.$$

Machines	M_1	M_2	M_3	M_4	$\cdots$	M_m
Jobs	m	$2m-1$	$2m-1$	$2m-2$	$\cdots$	$2m-\lfloor\frac{m}{2}\rfloor$
	m					
	m	$m+1$	$m+1$	$m+2$	$\cdots$	$2m-\lfloor\frac{m+1}{2}\rfloor$

Table 3.2: **Allocation of jobs to machines in OPT**

3.1.2 Bound LPT for uniform machines

In this section, we extend the previous result to a more general setting, where machines are allowed to operate at different speeds.

Proposition 3.2. *For scheduling m uniform machines, we have $\rho^{LPT} \leq 2$ for the deviation minimization problem.*

Proof. Assume to the contrary that there is an instance I such that

$$\begin{aligned} \rho^{\mathrm{LPT}}(I) &= \frac{\mathrm{dev}^{\mathrm{LPT}}(I) + T_0(I)}{\mathrm{dev}^*(I) + T_0(I)} > 2 \\ \Longrightarrow \mathrm{dev}^{\mathrm{LPT}}(I) - T_0(I) &> 2\mathrm{dev}^*(I) \geq 0. \end{aligned} \tag{3.12}$$

If $\mathrm{dev}^{\mathrm{LPT}}(I) = T_0(I) - T^{\mathrm{LPT}}_{\min}(I)$, then $T^{\mathrm{LPT}}_{\min}(I) < 0$ can be derived from (3.12), which is an obvious contradiction. Hence it must be the case that

$$\mathrm{dev}^{\mathrm{LPT}}(I) = T^{\mathrm{LPT}}_{\max}(I) - T_0(I).$$

Due to the definition of $\mathrm{dev}(I)$, for any arbitrary instance I, we have

$$\mathrm{dev}^*(I) + T_0(I) \geq T^*_{\max}(I).$$

Therefore,

$$\rho^{\mathrm{LPT}} = \frac{T^{\mathrm{LPT}}_{\max}(I)}{\mathrm{dev}^*(I) + T_0(I)} \leq \frac{T^{\mathrm{LPT}}_{\max}(I)}{T^*_{\max}(I)} \leq 2.$$

This completes the proof. □

The following example is to test the tightness of this bound.

Example 3.2. *Consider an instance I with two uniform machines with speed $s_1 = 1$ and $s_2 = a$ $(a > 1)$ respectively, and a list of jobs, the sizes of which satisfy $p_1 \geq \cdots \geq p_n$ and*

$$\sum_{i=1}^{n} \frac{p_i}{a} = (p_n - \varepsilon), \tag{3.13}$$

for $\varepsilon \to 0$. *Accordingly,* $T_0(I) = \frac{a}{a+1}(p_n - \varepsilon)$.

Due to (3.13), all the jobs will be assigned to the faster machine in LPT. Therefore,

$$T_1^{LPT}(I) = 0; \quad T_2^{LPT}(I) = \sum_i p_i/a = (p_n - \varepsilon).$$

However, in the optimal allocation, p_n *will be assigned to the slower machine, so that*

$$T_1^*(I) = p_n; \quad T_2^*(I) = \frac{a(p_n - \varepsilon) - p_n}{a}.$$

As a result, the performance of LPT on instance I *is calculated as*

$$\begin{aligned}\rho^{LPT}(I) &= \frac{dev^{LPT}(I) + T_0(I)}{dev^*(I) + T_0(I)} \\ &= \frac{\frac{2a}{a+1}(p_n - \varepsilon)}{p_n - \frac{a}{a+1}(p_n - \varepsilon) + \frac{a}{a+1}(p_n - \varepsilon)} = \frac{2a}{a+1} \cdot \frac{p_n - \varepsilon}{p_n}.\end{aligned}$$

Notice that $\rho^{LPT}(I) \to 2$ *as* $a \to \infty$ *and* $\varepsilon \to 0$.

3.1.3 Monotonicity for LPT

According to Theorem 2.1, in a truthful algorithm, the workload of each machine must remain non-increasing as its reporting speed decreases. The following example illustrates a case where a machine ends up with more jobs by reporting a slower speed, thus indicating that LPT cannot guarantee monotonicity.

Example 3.3. *Consider two machines with speeds* $s_1 = 10$ *and* $s_2 = 9$. *Let the job sequence be* $\{20, 11, 10\}$. *When applying LPT, the first job is assigned to* M_1, *followed by the second job assigned to* M_2. *The last job is also assigned to* M_2 *since* $\frac{20+10}{10} > \frac{11+10}{9}$.

Now assume s_1 *reduces to* 8, *while* s_2 *remains unchanged; thus* M_2 *becomes the faster machine. Applying LPT with the reduced speed set, the first job is now allocated to* M_2, *and the remaining two jobs are allocated to* M_1. *The workload of* M_1 *increases from* 20 *to* 21 *as its speed reduces, which implies LPT is manipulable.*

3.2 A quick result from makespan problem

Although the algorithm fails the monotonicity test, the study of LPT is instructive. It reveals the potentiality of using algorithms designed for the makespan problem to achieve good performance on the deviation problem. Now we present the general result of applying algorithms designed for the makespan minimization problem to the deviation minimization problem.

Theorem 3.3. *Let algorithm A be α-approximate for the makespan minimization problem.*

- *If $\alpha \leq 2$, then A is a 2-approximation algorithm for the deviation minimization problem;*
- *If $\alpha > 2$, then A is an α-approximation algorithm for the deviation minimization problem.*

Proof. Distinguish the optimal values derived under the deviation objective and the makespan objective with superscripts D^* and M^*, respectively. Assume that A is an α-approximation algorithm for the makespan minimization problem, i.e.,

$$\frac{T^A_{\max}(I)}{T^{M^*}_{\max}(I)} \leq \alpha. \tag{3.14}$$

When $\text{dev}^A(I) = T_0(I) - T^A_{\min}(I)$, we have

$$\rho^A(I) = \frac{2T_0(I) - T^A_{\min}(I)}{\text{dev}^{D^*}(I) + T_0(I)} \leq \frac{2T_0(I)}{T_0(I)} = 2.$$

If otherwise, $\text{dev}^A(I) = T^A_{\max}(I) - T_0(I)$, then

$$\rho^A(I) = \frac{T^A_{\max}(I)}{\text{dev}^{D^*}(I) + T_0(I)} \leq \frac{T^A_{\max}(I)}{T^{D^*}_{\max}(I)} \leq \frac{T^A_{\max}}{T^{M^*}_{\max}(I)} \leq \alpha.$$

The last inequality holds as a direct result from (3.14). This completes our proof. □

According to the theorem, any existing truthful mechanism for the makespan minimization problem can be directly implemented onto the deviation minimization problem and maintain the level of performance if its approximation ratio is above 2. In other words, when designing algorithms for the deviation minimization problem with approximation ratio bigger than 2, one only needs to focus on the maximum completion time. To further improve the performance, however, the algorithm de-

signer needs to concern both the maximum completion time and the minimum completion time at the same time.

Now we apply the theorem to the Monotone-RF algorithm proposed by Andelman et al. (2005), which is known to be truthful and 5-approximate for the makespan problem. The algorithm is inspired by the fractional assignment which considers each machine as a bin with bin size equal to the machine speed times a constant.

Definition 3.1. *A **fractional assignment** is a schedule obtained by breaking each job into pieces with sizes summing to the full size of that job, and assigning these pieces to machines so that all the machines can finish their assigned workloads at the same time.*

Monotone-RF

Input: a job sequence σ, and a non-increasing sorted speeds vector $s = (s_1, \ldots, s_m)$

Step 1. Set $t_1 = \frac{8}{5} s_1$.

Step 2. For $i \geq 2$, let t_i be the closest value of $\frac{s_1}{2.5^k}$ such that $t_i \leq s_i$, i.e. $k = \lfloor \log_{2.5} \frac{s_1}{s_i} \rfloor + 1$.

Step 3. Calculate the valid fractional assignment for the job sequence ε given the new speeds vector t. Let $T^f(t)$ be the value of the fractional solution.

Step 4. For each machine $i = 1, \ldots, m$, assign jobs in non-increasing order of job size to M_i (using speed t_i), until the completion time of M_i exceeds or equals threshold $T^f(t)$.

Step 5. Return the assignment.

Denote by T^f the constant derived from applying the fractional assignment. A fractional assignment is *valid* if for every piece assigned to M_i, the size of the job that the piece belongs is no more than its bin size $s_i \cdot T^f$. The smallest T^f for which there exists a valid fractional assignment is

$$T^f = \max_j \min_i \max \left\{ \frac{p_j}{s_i}, \frac{\sum_{k=1}^{j} p_k}{\sum_{l=1}^{i} s_l} \right\}. \tag{3.15}$$

The fractional assignment does not yield a truthful result with general speeds. However, after rounding machine speeds followed by rules defined in Monotone-RF, Andelman et al. (2005) prove the fractional assignment becomes monotone.

According to Theorem 3.3, Monotone-RF is a 5-approximate algorithm for the deviation minimization objective. Moreover, it remains monotone as it is a direct application of the algorithm. Finally we show that 5 is a tight bound.

Example 3.4. *Consider an instance with* 6 *machines and* 6 *jobs. Both machine speeds and job sizes are* $(1, 1, 1, 1-\varepsilon, 1-\varepsilon, 0.4)$*. The optimal solution for this instance is obvious: each machine receives a job with the same size as its speed. The maximum deviation* dev^* *equals* 0 *as a result.*

Now apply Monotone-RF to this instance. According to (3.15),

$$T^f(t) = \max\left\{\frac{0.4}{0.16}, \frac{5.4+\varepsilon}{3.36}\right\} = 2.5,$$

and the adjusted speeds vector t *is*

$$(1.6, 0.4, 0.4, 0.4, 0.4, 0.16).$$

Thus, the bin sizes of each machine become

$$(4, 1, 1, 1, 1, 0.4).$$

Monotone-RF only stops assigning jobs to machines when the workload of the current machine exceeds its bin size. Accordingly, the five biggest jobs are all assigned to the first machine, which leaves a deviation of $4-2\varepsilon$*. The approximation ratio, therefore, is* $5-2\varepsilon$*, which approaches to* 5 *when* $\varepsilon \to 0$*.*

3.3 LPT*

Using the same rounding skills, LPT can also become a truthful algorithm. The conjecture was made by Auletta et al. (2004), and was confirmed by Kovács (2005) with her adjusted LPT, or LPT*. In her work, Kovács (2005) shows that LPT* is truthful and yields 3-approximation ratio for the makespan problem. In the conclusion, she made a conjecture that the bound of LPT* can be further reduced to 8/3. In the following section, we show that a tighter bound is achievable regardless

of the false conjecture of 8/3.

LPT*

Input: a non-increasing sorted job sizes vector $p = (p_1, \ldots, p_n)$ and a non-increasing sorted machine speeds vector $s = (s_1, \ldots, s_m)$

Step 1. Round the machine speeds down to $t_i := 2^{\lfloor \log_2 s_i \rfloor}$, $1 \leq i \leq m$ (the rounded speeds remain ordered).

Step 2. Assign jobs in p to machines with the adjusted speeds based on LPT; tie is broken by assigning the job to the slowest machine with the smallest index.

Step 3. Among machines of the same rounded speed, reorder the assigned work such that $W_i \geq W_{i+1}$ holds.

Step 4. Return the assignment.

3.3.1 A counter example

We first show with a concrete example that breaks the conjecture of 8/3.

Example 3.5. *Consider an instance with* 21710 *machines, among which two machines have speed* 1023.84*, one's speed is* 64.08 *and the remaining* 21707 *machines have speed* 1.999*. The job set contains* 46347 *jobs, with sizes listed in the table in non-increasing order.*

No. of jobs	1085	2946	2946	2946	2946	2946	2946	2946
Job sizes	1.279	1.24	1.201	1.162	1.123	1.084	1.045	1.006
No. of jobs	2946	2946	2946	2946	2946	2946	2946	4018
Job sizes	0.993	0.954	0.915	0.876	0.837	0.798	0.759	0.72

Table 3.3: **Total jobs**

In the optimal solution, each of the fastest two machines with speed 1023.84 *receives* 1422 *jobs of size* 0.72*, and the machine with speed* 64.08 *receives* 89 *jobs with size* 0.72*. This gives each of them a completion time of* 1*. The allocations of the rest of the slow machines are summarized in Table 3.4, with the second column representing the size of each job assigned to a machine and the first column representing the number of such machines. For instance, the first row of the table says that there are* 1085 *machines of speed* 1.999 *assigning with two jobs, the sizes of which are* 1.279 *and* 0.72 *respectively. It is easy to calculate that each slow machine receives a total*

workload of 1.999*, equal to its machine speed. So far, we have finished the allocation of all jobs to machines and derived an optimal makespan of* 1*. Since the makespan of this allocation is exactly* T_0*, it is for sure that the makespan cannot be further improved.*

No. of machines	Job allocation
1085	$(1.279, 0.72)$
2946	$(1.24, 0.759)$
2946	$(1.201, 0.798)$
2946	$(1.162, 0.837)$
2946	$(1.123, 0.876)$
2946	$(1.084, 0.915)$
2946	$(1.045, 0.954)$
2946	$(1.006, 0.993)$

Table 3.4: **Job allocation on machines with speed** 1.999

The allocation under LPT is shown in Figure 3.1. Due to the large number of machines, those with the same job allocation are represented as one column with a number in bracket below showing how many of such machines exist. For example,* 1085 *machines with adjusted speed* 1 *are allocated two jobs with both sizes equal to* 0.993*. For the same reason, we did not list every single job allocated to the fast machines, but used the number in brackets to show how many times the job with the same size is assigned to the machine. The maximum completion time for this allocation is* $\frac{1.279\times63+0.993\times46+0.72\times62}{64.08} \approx 2.6669$*, which exceeds* $8/3 \approx 2.6667$*.*

Comparing the total workload assigned to each machine in OPT and in LPT* in the above counter example, it is obvious that all three fast machines receives more workload in LPT* than in OPT. Since the total workload should remain the same, the extra workload assigned to the fast machines in LPT* must be compensated by the slow machines. Suppose there is no restriction on the number of slow machines which recieve less workload in LPT* than in OPT, then the workload assigned to fast machines, and therefore, the makespan will increase without limit. However, in the following section, we will show case by case that the amount of workload that can be compensated by the slow machines is restricted by the fast machines themselves.

3.4 Improved upper bound for LPT*

In this section, we provide an improved upper bound for LPT*. Before the proof, we would like to highlight some frequently used properties of the LPT* algorithm,

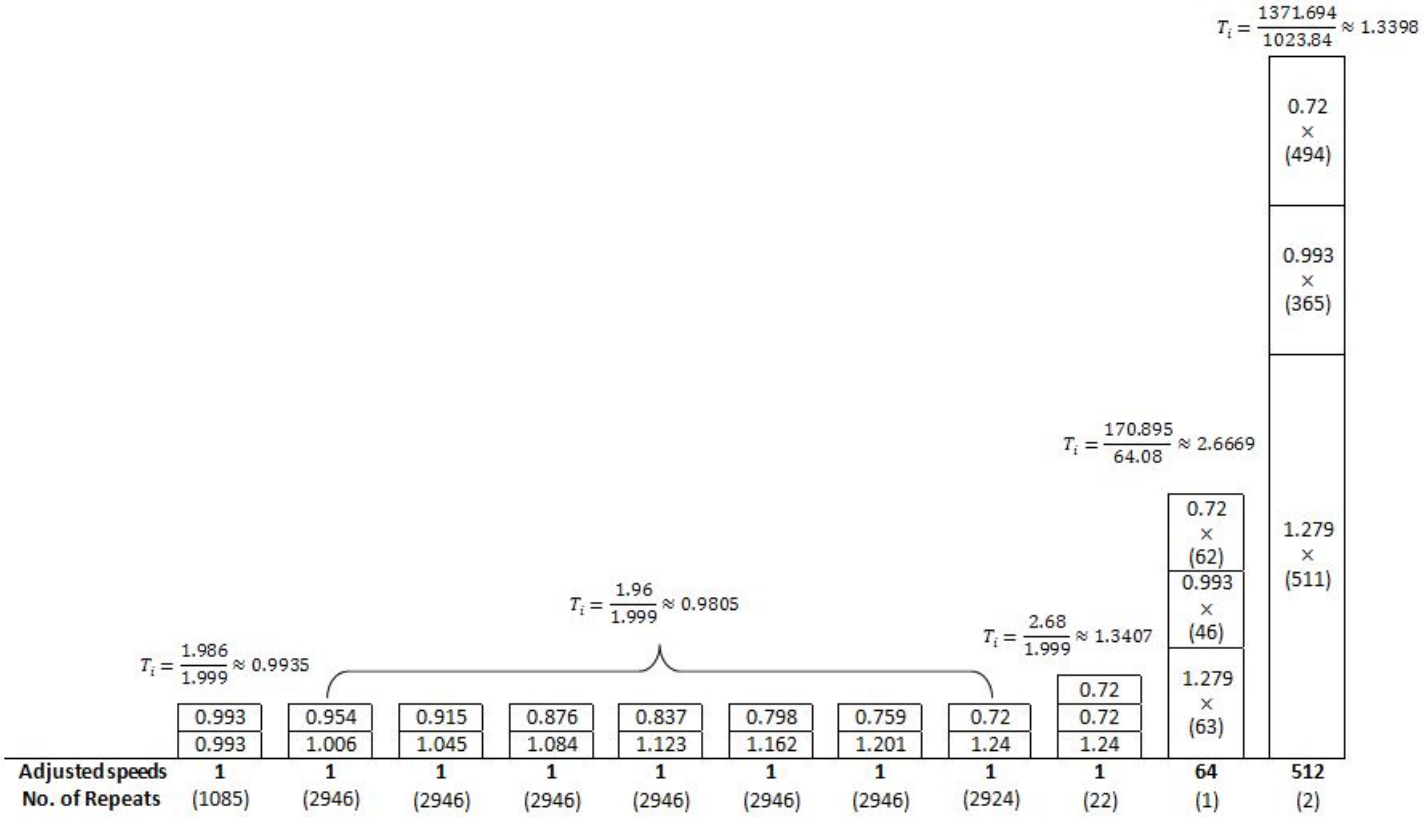


Figure 3.1: **Job Allocation in LPT***

which will facilitate the subsequent proofs.

Proposition 3.4 (Kovács, 2005)**.** *In LPT* scheduling the following hold:*

- *If $t_l = t_i/2$, then M_l receives its first job after the first job of M_i and before the second job of M_i.*
- *If $t_i = t_{i+1}$ and p_j is the first job assigned to M_i, then p_{j+1} is the first job assigned to M_{i+1}.*

In addition to the above property pointed out by Kovács (2005), the following feature holds for LPT*.

Proposition 3.5. *Let $t_l = 2^\eta t_i$ for some $\eta \geq 0$. In LPT* scheduling, M_l receives at most $2^\eta - 1$ jobs before the first job of M_i.*

Proof. Let p^* be the last job of M_l before the first job of M_i. Denote $W_i(p_j)$ the amount of work assigned to M_i right before p_j. According to LPT*, we derive

$$\frac{W_l(p^*) + p^*}{t_l} < \frac{p^*}{t_i} \implies W_l(p^*) + p^* < 2^\eta \cdot p^*.$$

Since the sizes of all the jobs assigned before p^* are at least p^*, $W_l(p^*) + p^*$ contains at most $2^\eta - 1$ jobs. □

Proposition 3.5 states that before the slow machine gets its first job, not too many jobs can be allocated to the fast machines in LPT*. This implies that although LPT* prioritizes the faster machines in assigning jobs, the priority is not unlimited.

3.4.1 Preliminaries

Some preliminary work and definitions are introduced in this part. Denote by $T^*_{\max}(I)$ the optimal makespan and $T_{\max}(I)$ the makespan generated by LPT* for instance I. Define $\theta(I) = T_{\max}(I)/T^*_{\max}(I)$ the approximation ratio we are going to bound for any instance I. The main body of the proof employs the minimum counter example approach. Let I^* denote an instance with the minimum number of jobs that satisfies $\theta(I^*) > 2.8$. Let k be the smallest machine index with the maximum completion time (i.e., the bottleneck machine in the makespan problem) in LPT* and p_n be the last job assigned to M_k. We delete all the jobs after p_n without decreasing $\theta(I^*)$; thus, p_n becomes the smallest job in the minimum counter-example. If not explicitly specified, the instance considered in the proofs is minimum counter example I^*. For simplicity, I^* is omitted if doing so causes no confusion.

Denote W_i and W_i^* the total amount of work assigned to M_i by LPT* and OPT respectively. Assume w.l.o.g. that $T^*_{\max} = 1$. Accordingly,

$$W_i^* \leq T^*_{\max} \cdot s_i < 2t_i, \qquad \forall\, M_i \in M. \tag{3.16}$$

Lemma 3.6. *There exists M_i, such that $W_i < 2t_i$.*

Proof. Assume by contradiction that $W_i \geq 2t_i$ for all machines, then the total amount of work assigned in LPT* is at least $2\sum_{M_i \in M} t_i$. Nevertheless, the total amount of work assigned in OPT is less than $2\sum_{M_i \in M} t_i$ according to (3.16). This is a contradiction since the total workload assigned in both schedules remain the same. Therefore, there exists at least one M_i, such that $W_i < 2t_i$. □

Lemma 3.6 allows to define h as the smallest index that satisfies

$$W_h < 2t_h \cdot T^*_{\max} = 2t_h. \tag{3.17}$$

Hence for all $i < h$,

$$W_i \geq 2t_i \cdot T^*_{\max} = 2t_i. \tag{3.18}$$

Let r be the largest index of machine with positive workload in LPT*, and assume that

$$t_h = 2^\beta t_r, \tag{3.19}$$

where β is some integer. Assume that $t_r = 1$. This can be achieved by dividing each original speed by the smallest non-empty adjusted speed. Notice that this assumption will not lose generality of our proof, because machine speeds and job sizes can be adjusted proportionally so that $1 \leq s_r < 2$, and $T^*_{\max} = 1$ remains unchanged. We illustrate the generality of this assumption with an example.

Example 3.6. *Consider a set of speeds $s = \{27, 16, 13, 5\}$, the adjusted speeds of which are $t = \{16, 16, 8, 4\}$. Assume that when allocating a set of jobs on these machines, the slowest machine is non-empty in LPT*; thus, $t_r = 4$ in this case. Dividing each original speed by 4 derives $s' = \{6.75, 4, 3.25, 1.25\}$. Since machine speeds reduce proportionally, the optimal solution remains unchanged. The adjusted speeds for s' are $t' = \{4, 4, 2, 1\}$, which also reduced proportionally compared to t; as a result, the allocation under LPT* also remains unchanged. Finally, the job size is adjusted according to s', so that $T^*_{\max} = 1$.*

A machine speed can be smaller than 1, but according to our assumption such a

machine is empty (i.e., with zero workload) in LPT*. The following example is used to illustrate h, r and β.

Example 3.7. *Consider an instance with* 20 *machines. The speeds of the five fastest machines are* 15.1, 6.9, 6.8, 3.9 *and* 3.9 *and the rests are* 1.9. *Jobs needed to be allocated are:* 15 *jobs with sizes* 1.9 *and* 1.4, *respectively, and* 12 *jobs with size* 1.3. *The average completion time is* $T_0 = 1$. *In the optimal solution, each machine receives a workload equal to its machine speed, therefore* $T^*_{\max} = 1$, *which complies with our assumption.*

	1.3							
	1.3							
	1.3							
	1.4							
	1.4							
	1.4							
	1.4	1.3	1.3					
	1.4	1.3	1.3					
	1.4	1.4	1.4	1.3	1.3	15		
	1.4	1.4	1.4	1.3	1.3			
	1.4	1.4	1.4	1.3	1.3	1.9	⋯	1.9
Machine speeds	15.1	6.9	6.8	3.9	3.9	1.9	⋯	19

Table 3.5: **Job allocation in OPT**

Now allocate jobs to machines following LPT. The result is shown in Table 3.6. The resulting workload of each machine is presented in the last row. As can be seen, the workloads of the first five machines all exceed two times their adjusted speeds. The maximum completion time* $\frac{18.5}{15.1} \approx 1.225$ *is achieved on the fastest machine. Hence,* $h = 6$, $r = 20$, $k = 1$, *and* $\beta = 0$.

Next let us consider some characteristics of I^*. Let $s_k = \alpha t_k$ for some $1 \leq \alpha < 2$. In LPT*, we have

$$T_{\max} = \frac{W_k}{s_k} = \frac{W_k}{\alpha \cdot t_k} = \theta \geq 2.8 \implies W_k = \alpha \cdot \theta \cdot t_k \tag{3.20}$$

for the bottleneck machine M_k, and

$$\frac{W_i + p_n}{t_i} \geq \alpha \cdot \theta \implies W_i \geq \alpha \cdot \theta \cdot t_i - p_n \tag{3.21}$$

for a non-bottleneck machine M_i.

	1.3							
	1.3							
	1.3							
	1.3							
	1.9							
	1.9							
	1.9	1.3	1.3					
	1.9	1.3	1.3					
	1.9	1.9	1.9	1.3	1.3	15		
	1.9	1.9	1.9	1.3	1.3			
	1.9	1.9	1.9	1.9	1.9	1.4	$\cdots$	1.4
Machine speeds	8	4	4	2	2	1	$\cdots$	1
Workload	18.5	8.3	8.3	4.5	4.5	1.4	$\cdots$	14

Table 3.6: **Job allocation in LPT***

Lemma 3.7. *If a machine is empty in LPT*, then it remains empty in OPT.*

Proof. Let M_i be an empty machine in LPT*, i.e., $W_i = 0$. According to (3.21), we derive

$$\frac{p_n}{t_i} > \theta \implies p_n > 2t_i.$$

However, since $T^*_{\max} = 1$, $W^*_i < 2t_i$, which implies that the biggest job assigned to M_i in OPT is smaller than p_n. Notice that all jobs smaller than p_n are deleted from an instance. Therefore, M_i remains empty in OPT. □

Corollary 3.8. $W_h > 0$ *and* $\beta \geq 0$.

Proof. Assume to the contrary that $W_h = 0$, then any machine after h is also empty in LPT*. According to Lemma 3.7, $W^*_i = 0$ for all $i \geq h$. On the other hand, due to the definition of M_h, we have

$$\sum_{i<h} W_i \geq 2 \sum_{i<h} t_i > \sum_{i<h} W^*_i,$$

which implies that the amount of workload assigned in LPT* exceeds that in OPT, and that gives a contradiction.

According to the definition of r, we have $h \leq r$, i.e., $t_h \geq t_r$, which implies that $\beta \geq 0$. □

Let $i = h$ in (3.21), we derive

$$p_n > 2^\beta \cdot \theta - W_h > 2^\beta \cdot \theta - 2t_h = (\theta - 2) \cdot 2^\beta.$$

Since $W_h > 0$, thus $W_h \geq p_n$, which in turn implies

$$(\theta - 2) \cdot 2^\beta < p_n < 2^{\beta+1}. \tag{3.22}$$

Finally, we provide the following definitions that are often referred to in the proofs before presenting the formal proof.

Definition 3.2. *M_i is a **taker** if $W_i > W_i^*$; M_i is a **giver** if $W_i < W_i^*$; M_i is a **deadwood** if $W_i = W_i^*$.*

We indicate these three types of machines with a concrete example.

Example 3.8. *Consider the following example, which contains* 6 *machines with speeds*

$$(14, 3.9, 3.9, 3.9, 3.9, 1).$$

We assign a total number of 9 *jobs to these machines, of which* 4 *have size* 3.9 *and* 5 *have size* 2.8*. In the optimal solution, apart from the slowest machine, each machine is assigned a workload equal to its speed.*

	2.8					
	2.8					
	2.8					
	2.8					
	2.8	3.9	3.9	3.9	3.9	
Machine speeds	14	3.9	3.9	3.9	3.9	1

Table 3.7: **Job allocation in OPT**

In LPT, the slowest machine remains slack, while the workload on the fastest machine increased by* $(3.9 - 2.8) \times 3 = 3.3$*, which leads to the makespan increasing to* $\frac{14+3.3}{14} \approx 1.24$*.*

In the above example, M_1 is a taker, M_3 to M_5 are the givers, while M_2 and M_6 are deadwood. The approximation ratio derived from the example instance is quite low. This is because with only three givers in the instance, the workload contributed by the givers is not sufficient to satisfy the workload required by (3.20) and (3.21) if θ derives a larger value.

	2.8					
	2.8					
	3.9					
	3.9					
	3.9	3.9	2.8	2.8	2.8	
Machine speeds	8	2	2	2	2	1

Table 3.8: **Job allocation in LPT***

Definition 3.3. *A series of sets of machines and their corresponding notations are defined as follows:*

- *Set of* 1*-machines:* $S_1 = \{M_i | t_i = 1\}$*;*

- *Set of* $\tilde{1}$*-machines:* $S_{>1} = \{M_i | t_i > 1\}$*;*

- *Set of* 2*-machines:* $S_2 = \{M_i | t_i = 2\}$*;*

- *Set of* $\tilde{2}$*-machines:* $S_{>2} = \{M_i | t_i > 2\}$*;*

- *Set of non-empty machines:* $M = \{M_i | W_i > 0\}$.

3.4.2 An easy case

An easy parametric bound of LPT* is provided for a general instance I.

Proposition 3.9. $\theta(I) \leq 2 + \frac{2}{2^\beta - 1}$ *for* $\beta \geq 1$.

Proof. Let p^* be the first job of M_r in LPT*, then

$$\frac{W_h(p^*) + p^*}{t_h} \geq \frac{p^*}{t_r}.$$

Together with (3.17) and (3.22), we derive

$$\begin{aligned} 2^{\beta+1} = 2t_h > & W_h(p^*) \geq p^* \cdot (t_h - 1) \geq p_n \cdot (t_h - 1) > 2^\beta(\theta(I) - 2)(2^\beta - 1) \\ & \implies 2 > (\theta(I) - 2)(2^\beta - 1) \\ & \overset{\beta \geq 1}{\implies} \theta(I) < 2 + \frac{2}{2^\beta - 1}. \end{aligned}$$

□

The parametric bound only depends on β, and therefore is easy to compute. It

provides a much more accurate bound for LPT* particularly when β is big. Additionally, Proposition 3.9 states that $\theta < \frac{8}{3}$ for $\beta \geq 2$. On the other hand, β is a non-negative integer. Thus we only need to consider the bound of LPT* for $\beta \in \{0, 1\}$.

3.4.3 $\beta = 0$ and OPT assigns one job to each 1-machine

When $\beta = 0$, we derive $t_h = 1$ from (3.19), which implies

$$W_i > 2t_i > W_i^*, \ \forall \ M_i \in S_{>1}$$

according to the definition of M_h. In other words, all $\tilde{1}$-machines are takers, which leaves 1-machines to be the only possible givers.

For any feasible solution, the following equation must hold:

$$\sum_{t_i>1}(W_i - W_i^*) = \sum_{t_i=1}(W_i^* - W_i). \tag{3.23}$$

This is because the total workload assigned in OPT equals to that in LPT* for a feasible solution. In what follows, we show that (3.23) will be violated. The left-hand side of (3.23) presents the minimum amount of workload needed to be compensated by the entire set of givers. We first derive a lower bound for it, i.e.,

$$\min_{\pi}\left\{\sum_{t_i>1}(W_i - W_i^*)\right\} \geq \text{LB}.$$

Then we show that the maximum workload can be compensated by the givers, i.e., the upper bound for the right-hand side is strictly smaller than LB. In mathematical terms,

$$\max_{\pi}\left\{\sum_{t_i=1}(W_i^* - W_i)\right\} \leq \text{UB} < \text{LB}.$$

Notice that for $t_i \geq 2$, W_i is bounded by the bigger value between (3.21) and (3.18). While $\alpha \cdot \theta \cdot t_i - p_n > 2t_i$ for $t_i \geq 2$. In other words, W_i is bounded by (3.21) for $t_i \geq 2$ in (3.23). As a result, we derive

$$\sum_{t_i>1}(W_i - W_i^*) > \sum_{t_i>1}(\theta t_i - p_n - W_i^*) = \text{LB}. \tag{3.24}$$

In fact, LB can be written as a function of the adjusted speeds of all the $\tilde{1}$-machines due to (3.16). Once $S_{>1}$ is fixed (in an arbitrary way), LB becomes a fixed value which can be reached when $s_i = W_i \to 2t_i$, $\forall\ M_i \in S_{>1}$.

Next, we show that for each fixed $S_{>1}$, there exists an instance that maximizes UB. We call such an instance the *best case*. A best case instance has the following properties:

1. The number of 1-machines, denoted by ℓ, is no less than $\sum_{t_i>1}(t_i - 1)$;
2. In OPT, the length of jobs assigned to 1-machines are no less than the length of jobs assigned to $\tilde{1}$-machines;
3. The first largest $\sum_{t_i>1}(t_i - 1)$ jobs are assigned before LPT* assigns the first job to a 1-machine.

The properties do not contradict to each other, meaning that they can co-exist in the same instance. Now we show that the best case allows UB to be maximized.

Lemma 3.10 (Epstein et al., 2013)**.** *Assume that $s_1 \le s_2 \le \cdots \le s_m$. There exists an optimal schedule π for the makespan minimization problem which satisfies $W_1(\pi) \le W_2(\pi) \le \cdots \le W_m(\pi)$.*

Renumber all the 1-machines so that $s_1 \ge s_2 \ge \cdots \ge s_\ell$. Denote by q_i^* the unique job assigned to 1-machine M_i in OPT. According to Lemma 3.10, there exists an optimal solution such that $q_1^* \ge q_2^* \ge \cdots \ge q_\ell^*$. Define $q_0^* = 2$. Denote by q_i the first job assigned to 1-machine M_i in LPT*.

Claim 3.1. *For $1 \le j \le \ell$ and $j' \le j$, if $q_{j'} \in [q_j^*, q_{j-1}^*)$, then $W_{j'+a} \ge q_{j+a}^*$, for $a = 0, \ldots, \ell - j$.*

Proof. Since $q_{j'} \ge q_j^*$, q_{j+1}^* has not yet been allocated after 1-machine j' gets its first job; thus the next available job to be assigned is at least as big as q_{j+1}^*. On the other hand, LPT* will assign the next available job to 2-machine $j' + 1$. Together we derive that $q_{j'+1} \ge q_{j+1}^*$. Similarly, we have $q_{j'+2} \ge q_{j+2}^*$ and so on, until $q_{j'+\ell-j} \ge q_\ell^*$. Since $W_{j'+a} \ge q_{j'+a}$, we derive $W_{j'+a} \ge q_{j+a}^*$, for $a = 0, \ldots, \ell - j$. □

Let $j' = 1$, we derive a special case for Claim 3.1.

Claim 3.2. *If $q_1 \in [q_j^*, q_{j-1}^*)$ for $j = 1, \ldots, \ell$, then $W_{1+a} \ge q_{j+a}^*$, for $a = 0, \ldots, \ell - j$.*

Claim 3.3. *If $q_1 \in [q_j^*, q_{j-1}^*)$ for $j = 1, \ldots, \ell$, then $j - 1 \le \sum_{t_i>1}(t_i - 1)$.*

Proof. According to the sequence of LPT*, all jobs that are bigger than or equal to q^*_{j-1} are assigned before q_1, which is the first job that LPT* assigns to a 1-machine. According to the assumption of $q^*_1 \geq q^*_2 \geq \cdots \geq q^*_{j-1}$, there are at least $j-1$ such jobs assigned before q_1. On the other hand, according to Proposition 3.5, each $\tilde{1}$-machine i receives at most $t_i - 1$ jobs before the first job of a 1-machine. This implies that the total number of jobs that can be assigned before q_ℓ is at most $\sum_{t_i>1}(t_i-1)$. Together we have $j-1 \leq \sum_{t_i>1}(t_i-1)$. $\square$

Subsequently, let us consider the right-hand side of equation (3.23). According to Claim 3.2, for $q_1 \in [q^*_j, q^*_{j-1})$ $(1 \leq j \leq \ell)$, the right-hand side of (3.23) becomes

$$\begin{aligned}
\sum_{t_i=1}(W^*_i - W_i) &= \sum_{i=1}^{\ell}(q^*_i - W_i) \\
&= \sum_{i=1}^{j-1} q^*_i + q^*_j + \cdots + q^*_\ell - W_1 - \cdots - W_{\ell-j+1} - \sum_{i=1}^{j-1} W_{\ell-j+1+i} \\
&\leq \sum_{i=1}^{j-1} q^*_i + q^*_j + \cdots + q^*_\ell - q^*_j - \cdots - q^*_\ell - \sum_{i=1}^{j-1} W_{\ell-j+1+i} \qquad (3.25) \\
&= \sum_{i=1}^{j-1}(q^*_i - W_{\ell-j+1+i}).
\end{aligned}$$

For each $M_i \in S_1$, we have $q^*_i < 2$ and $W_i \geq f$, where $f = \max\{\theta - p_n, p_n\}$. Together with Claim 3.3, we derive

$$\sum_{t_i=1}(W^*_i - W_i) \leq \sum_{i=1}^{j-1}(q^*_i - W_{\ell-j+1+i}) < \sum_{t_i>1}(t_i-1)(2-f) = \text{UB}. \qquad (3.26)$$

According to (3.25), UB is maximized when $j-1$ attains its largest possible value, i.e., $j-1 = \sum_{t_i>1}(t_i-1)$. In other words, LPT* assigns the largest $\sum_{t_i>1}(t_i-1)$ jobs from q^*_i to $\tilde{1}$-machines before it assigns the first job to a 1-machine. These together indicate Properties 1 and 3. Furthermore, (3.26) implies that UB is maximized when $W_{1+a} = q^*_{j+a}$, for $a = 0, \ldots, \ell - j$. This indicates that $q^*_j, q^*_{j+1}, \ldots, q^*_\ell$ are assigned to the first $\ell - j + 1$ 1-machines in LPT*, and therefore Property 2 is true.

Notice that for each item in (3.26), there is a corresponding item in (3.24). Denote by LB_i and UB_i the items associated to t_i in (3.24) and (3.26) respectively. Let us compare the values of LB and UB pairwise. For each $t_i \geq 4$, when $p_n \geq \frac{\theta}{2}$,

$f = p_n$. Together with (3.16), we have

$$\begin{aligned}\mathrm{LB}_i - \mathrm{UB}_i =&(\theta t_i - p_n - W_i^*) - (t_i - 1) \cdot (2 - f)\\ >&(\theta t_i - p_n - 2t_i) - (t_i - 1) \cdot (2 - p_n)\\ =&(\theta - 4 + p_n)t_i + 2 - 2p_n\\ \geq&2p_n + 4\theta - 14 > 5\theta - 14 \geq 0.\end{aligned}$$

When $p_n < \frac{\theta}{2}$, $f = \theta - p_n$, and similarly

$$\begin{aligned}\mathrm{LB}_i - \mathrm{UB}_i =&(\theta t_i - p_n - W_i^*) - (t_i - 1) \cdot (2 - f)\\ >&(\theta t_i - p_n - 2t_i) - (t_i - 1) \cdot (2 - \theta + p_n)\\ =&(2\theta - 4 - p_n)t_i + 2 - \theta\\ \geq&7\theta - 14 - 4p_n > 5\theta - 14 \geq 0.\end{aligned}$$

Finally, for each $t_i = 2$, $t_i - 1 = 1$. In other words, there is a unique 1-machine in each of the corresponding items in (3.26). The best case requires a job of size $\tilde{p} \geq \theta - p_n$ to be assigned to this 1-machine in LPT*. Moreover, $\tilde{p}$ is assigned to M_i in OPT. This is because in the proof for $t_{i'} \geq 4$, it is assumed that the entire workload assigned to $M_{i'}$ in OPT has been distributed between $M_{i'}$ and its corresponding $t_{i'} - 1$ 1-machines, with no extra workload left to compensate other pairs.

When $1.2 < p_n < 2$, we show that M_i contains at most two jobs in OPT. Assume to the contrary that there are more than two jobs on M_i in OPT, then after taking $\tilde{p}$ from W_i^*, the remaining workload should be at least $2p_n$. Since $W_i^* < 4$ for $t_i = 2$, we derive $2p_n \leq W_i^* - \tilde{p} < W_i^* - \theta + p_n \implies p_n \leq 1.2$, which contradicts $p_n > 1.2$. According to the job allocation sequence in LPT*, $\tilde{p}$ is the bigger one, and therefore $W_i^* \leq 2\tilde{p}$. Thus,

$$\begin{aligned}\mathrm{LB}_i - \mathrm{UB}_i =&(\theta t_i - p_n - W_i^*) - (t_i - 1) \cdot (2 - \tilde{p})\\ \geq&2\theta - p_n - 2\tilde{p} - 2 + \tilde{p} \geq 2.2 - \tilde{p} > 0.\end{aligned}$$

When $p_n \leq 1.2$,

$$\begin{aligned}\mathrm{LB}_i - \mathrm{UB}_i =&(\theta t_i - p_n - W_i^*) - (t_i - 1) \cdot (2 - \tilde{p})\\ \geq&2\theta - p_n - 4 - 2 + \theta - p_n > 2.4 - 2\tilde{p} \geq 0.\end{aligned}$$

To sum up, we show that each item in (3.26) is strictly smaller than the correspond-

ing item in (3.24). Thus, we prove LB > UB.

3.4.4 $\beta = 0$ and OPT assigns two jobs to some 1-machines

If there exists a 1-machine that receives two jobs in OPT, then $p_n < 1$. Additionally, if 1-machine M_i receives more than two jobs in LPT*, then it becomes a taker since $W_i \geq 3p_n > 2 > W_i^*$. There is no need to consider the case where a taker 1-machine exists, because it will only reduce the value of $\sum_{t_i=1}(W_i^* - W_i)$.

Denote by ℓ_0 and ℓ_1 respectively, the number of 1-machines that contain only 1 job in OPT and LPT*. The entire set of jobs assigned to 1-machines in OPT can be presented as $q_1^* \geq q_2^* \geq \cdots \geq q_{2\ell-\ell_0}^*$. Denote by $N \leq \sum_{t_i>1}(t_i - 1)$ the number of jobs assigned by LPT* before the first job of a 1-machine. Since $W_i^* \to 2$ for all 1-machines is a condition for maximizing $\sum_{t_i=1}(W_i^* - W_i)$, we are able to construct a job allocation in OPT as shown in Table 3.9. We consider the cases for $\ell_1 = 0$ and $\ell_1 > 0$ in turn.

M_1	M_2	$\cdots$	M_{ℓ_0}	M_{ℓ_0+1}	M_{ℓ_0+2}	$\cdots$	M_ℓ
q_1^*	q_2^*	$\cdots$	$q_{\ell_0}^*$	$q_{\ell_0+1}^*$	$q_{\ell_0+2}^*$	$\cdots$	q_ℓ^*
				$q_{2\ell-\ell_0}^*$	$q_{2\ell-\ell_0-1}^*$	$\cdots$	$q_{\ell+1}^*$

Table 3.9: **Job allocation on 1-machines in OPT**

When $\ell_1 = 0$, all the giver 1-machines receive two jobs in LPT*, which implies that $\ell_0 \leq N$. One additional property is added on top of the previous properties of a best case, which requires $\ell_0 = 0$.

Now assign jobs to machines under LPT*. According to the best case, jobs assigned to 1-machines in OPT will be the first jobs to assign. The first N jobs from q_1^* to q_N^* are assigned to $\tilde{1}$-machines, followed by the next ℓ jobs, from q_{N+1}^* to $q_{N+\ell}^*$, each being assigned to a 1-machine. Among the remaining $\ell - \ell_0 - N$ jobs from $q_{N+\ell+1}^*$ to $q_{2\ell-\ell_0}^*$, assume that N_1 are assigned to $\tilde{1}$-machines, and the rest $\ell - \ell_0 - N - N_1$ jobs, marked as $q_1, \ldots, q_{\ell-\ell_0-N-N_1}$, are assigned to 1-machines. Further assume that among the N_1 jobs assigned to $\tilde{1}$-machines, N_2, marked as $q'_1, \ldots, q'_{N_2}$, are assigned to M_i, for $i = \ell_0 + 1, \ldots, N$, in OPT. The remaining $N_1 - N_2$ jobs, marked as $q''_1, \ldots, q''_{N_1-N_2}$, are assigned to M_i, for $i = N + 1, \ldots, \ell$, in OPT.

Since each 1-machine receives two jobs, LPT* still needs another $\ell_0 + N + N_1$ jobs to occupy the remaining 1-machines. Denote these jobs as $p_1, \ldots, p_{\ell_0+N+N_1}$.

To sum up, the total 2ℓ jobs assigned to 1-machines by LPT* are: $q^*_{N+1}, \ldots, q^*_{N+\ell}$, $q_1, \ldots, q_{\ell-\ell_0-N-N_1}$, and $p_1, \ldots, p_{\ell_0+N+N_1}$. Together with $p_i \geq p_n$, we derive

$$\begin{aligned}
\sum_{t_i=1}(W^*_i - W_i) &= \left(\sum_{i=1}^{\ell_0} q^*_i + \sum_{i=\ell_0+1}^{N} q^*_i + \sum_{i=N+1}^{N+\ell} q^*_i + \sum_{i=N+\ell+1}^{2\ell-\ell_0} q^*_i\right) \\
&\quad - \left(\sum_{i=N+1}^{N+\ell} q^*_i + \sum_{i=1}^{\ell-\ell_0-N-N_1} q_i + \sum_{i=1}^{\ell_0+N+N_1} p_i\right) \\
&\leq \left(\sum_{i=1}^{\ell_0} q^*_i + \sum_{i=\ell_0+1}^{N} q^*_i + \sum_{i=N+1}^{N+\ell} q^*_i + \sum_{i=N+\ell+1}^{2\ell-\ell_0} q^*_i\right) \\
&\quad - \left(\sum_{i=N+1}^{N+\ell} q^*_i + \sum_{i=1}^{\ell-\ell_0-N-N_1} q_i + (\ell_0+N+N_1)p_n\right).
\end{aligned}$$

Now we are going to derive the upper bound of the above equation. Notice that if a job is assigned to a 1-machine in both LPT* and OPT, then this job will be canceled out in the above inequalities, and therefore, has no influence on the upper bound. Therefore, UB only depends on jobs that are assigned to different parties in OPT and LPT*. These include the $N + N_1$ jobs that are assigned to 1-machines in OPT yet being assigned to $\tilde{1}$-machines in LPT*, and $\ell_0 + N + N_1$ jobs that are assigned to $\tilde{1}$-machines in OPT yet being assigned to 1-machines in LPT*. We classify these jobs into groups, and discuss the upper bound for each group.

For each $i = 1, \ldots, \ell_0$, we have

$$q^*_i - 2p_n < 2 - 2p_n. \tag{3.27}$$

For each q'_i $(i = 1, \ldots, N_2)$, it is possible to find a corresponding job from $q^*_{i'}$ $(i' = \ell_0 + 1, \ldots, N)$, so that q'_i and $q^*_{i'}$ are assigned to the same 1-machine in OPT. Thus,

$$q'_i + q^*_{i'} - 2p_n < 2 - 2p_n. \tag{3.28}$$

For the rest $N - \ell_0 - N_2$ jobs in q^*_i $(i = \ell_0 + 1, \ldots, N)$, we have

$$q^*_i - p_n < 2 - p_n - p_n = 2 - 2p_n. \tag{3.29}$$

Finally, for each q''_i $(i = 1, \ldots, N_1 - N_2)$, it is possible to find a corresponding job from $q^*_{i'}$ $(i' = N + 1, \ldots, N + \ell)$, so that q'_i and $q^*_{i'}$ are assigned to the same 1-machine in OPT. Additionally, $q^*_{i'}$ is among the first ℓ jobs assigned to the 1-machines in LPT*,

therefore $q^*_{i'} + p_n \geq f$, where $f = \max\{\theta - p_n, 2p_n\}$. Thus,

$$q'_i + q^*_{i'} - (q^*_{i'} + p_n) = q'_i - p_n < 2 - f. \tag{3.30}$$

From the combination of (3.27), (3.28), (3.29), and (3.30), we derive

$$\begin{aligned}\sum_{t_i=1}(W^*_i - W_i) <& \ell_0(2-2p_n) + N_2(2-2p_n) + (N-\ell_0-N_2)(2-2p_n) \\ &+ (N_1 - N_2)(2-f) = N(2-2p_n) + (N_1-N_2)(2-f) = \text{UB}.\end{aligned}$$

It is clear that UB is independent of ℓ_0, therefore let $\ell_0 = 0$ for simplicity. Since $2-f > 0$, to maximize UB, we shall let $N_2 = 0$. In addition, due to $2-2p_n \geq 2-f$, we shall let N attain its maximum possible number, which is $\sum_{t_i>1}(t_i-1)$. Finally, since each of the first N jobs assigned to $\tilde{1}$-machines will increase UB by $q^*_i - p_n < 2 - p_n - p_n$, for $i \in \{1, \ldots, N\}$, we let the length of these N jobs be as big as possible.

If UB $\geq$ LB, then according to (3.24), we have

$$\begin{aligned} & N(2-2p_n)+(N_1-N_2)(2-f) \geq \sum_{t_i>1}(\theta t_i - p_n - 2t_i) \\ \Longrightarrow \quad & N_1(2-f) \geq \sum_{t_i>1}(\theta t_i - p_n - 2t_i) - \sum_{t_i>1}(t_i-1)(2-2p_n) \\ & = \sum_{t_i>1}(\theta t_i - 4t_i + 2t_i p_n + 2 - 3p_n). \end{aligned} \tag{3.31}$$

In what follows, we show that (3.31) will be violated, thus we derive UB < LB.

Claim 3.4. *$q^*_i < 2 - p_n$, for $i = 1, \ldots, 2\ell$.*

Proof. Assume to the contrary that there exists $q^*_i \geq 2 - p_n$. Since $\ell_0 = 0$, all 1-machines receive two jobs in OPT. Notice that q^*_i is assigned to some 1-machine $M_{i'}$ in OPT. Thus for $M_{i'}$, we have $W^*_{i'} \geq 2 - p_n + p_n = 2$, contradicting to (3.16). □

Claim 3.5. *Before LPT* assigns its first 2ℓ jobs (i.e., $q^*_i, i = 1, \ldots, 2\ell$), the makespan in the best case will not exceed 2.*

Proof. In the best case, the largest $N = \sum_{t_i>1}(t_i - 1)$ jobs, i.e., $q^*_1, \ldots, q^*_N$, are assigned to the $\tilde{1}$-machines first. Hence before the first job on a 1-machine, each

$\tilde{1}$-machine M_i receives $t_i - 1$ jobs with sizes no bigger than $2 - p_n$, i.e.,

$$\frac{W_i(q^*_{N+1})}{t_i} \leq \frac{(t_i - 1)(2 - p_n)}{t_i} < 2, \forall\ M_i \in S_{>1},$$

where $W_i(p_j)$ denotes the total amount of workload on M_i right before the assignment of p_j as defined previously. Subsequently, jobs $q^*_{N+1}, \ldots, q^*_{N+\ell}$ are assigned to 1-machines. Accordingly, job assignments on 1-machines in LPT* are as below.

M_1	M_2	$\cdots$	$M_{\ell-N}$	$M_{\ell-N+1}$	$\cdots$	M_ℓ
q^*_{N+1}	q^*_{N+2}	$\cdots$	q^*_ℓ	$q^*_{\ell+1}$	$\cdots$	$q^*_{N+\ell}$
		$\cdots$			$\cdots$	

Table 3.10: **Job allocation on 1-machines in LPT***

Since $q^*_i < 2 - p_n$, up to this point, the completion times on each 1-machine are strictly smaller than 2. Next we will show that for the last $\ell - N$ jobs from q^*_i, none of them can be assigned to a machine with a resulting completion time greater than 2.

Before assigning $q^*_{N+\ell+1}$, there is only one job on M_ℓ, and $W_\ell = q^*_{N+\ell}$. Therefore,

$$\frac{W_\ell(q^*_{N+\ell+1}) + q^*_{N+\ell+1}}{1} = q^*_{N+\ell} + q^*_{N+\ell+1} < 2.$$

If $q^*_{N+\ell+1}$ is not assigned to M_ℓ, then it must be assigned to another machine which completes it earlier. Similarly, before assigning q^*_{N+i}, $i \in \{\ell + 1, \ldots, 2\ell - N\}$, there is only one job on $M_{2\ell+1-i}$, and $W_{2\ell+1-i} = q^*_{N+2\ell+1-i}$. Therefore,

$$\frac{W_{2\ell+1-i}(q^*_{N+i}) + q^*_{N+i}}{1} = q^*_{N+2\ell+1-i} + q^*_{N+i} < 2,$$

which implies that q^*_{N+i} must be assigned to some machine that completes it no later than $M_{2\ell+1-i}$ does. To sum up, all the jobs that are assigned to 1-machines in OPT will be processed within a completion time less than 2. This completes our proof. □

Denote $\bar{p}$ the average size of the N_1 jobs assigned to $\tilde{1}$-machines. According to Claim 3.5,

$$N(2 - p_n) + N_1\bar{p} < \sum_{t_i > 1} 2t_i \implies N_1 < \frac{\sum_{t_i>1}(p_n t_i + 2 - p_n)}{\bar{p}}. \tag{3.32}$$

Carrying (3.32) into (3.31) we derive

$$\begin{aligned}
&\frac{\sum_{t_i>1}(p_n t_i + 2 - p_n)}{\bar{p}} \cdot (2-f) > \sum_{t_i>1}(\theta t_i - 4t_i + 2t_i p_n + 2 - 3p_n) \\
\Longrightarrow \quad &\sum_{t_i>1}(p_n t_i + 2 - p_n) \cdot (2-f) > \sum_{t_i>1}(\theta t_i - 4t_i + 2t_i p_n + 2 - 3p_n) \cdot \bar{p} \\
&\geq \sum_{t_i>1}(\theta t_i - 4t_i + 2t_i p_n + 2 - 3p_n) \cdot p_n. \quad (3.33)
\end{aligned}$$

When $2p_n \geq \theta - p_n$, bringing $f = 2p_n$ into (3.33), we derive

$$\sum_{t_i>1}\left((4p_n^2 - 6p_n + \theta p_n)t_i - 5p_n^2 + 8p_n - 4\right) \leq 0.$$

However, since $t_i \geq 2$ and $4p_n^2 - 6p_n + \theta p_n > 0$, we have

$$\begin{aligned}
&(4p_n^2 - 6p_n + \theta p_n)t_i - 5p_n^2 + 8p_n - 4 \\
\geq &8p_n^2 - 12p_n + 2\theta p_n - 5p_n^2 + 8p_n - 4 \\
= &3p_n^2 - 4p_n + 2\theta p_n - 4 > 0.
\end{aligned}$$

which is a clear contradiction.

When $2p_n < \theta - p_n$, bringing $f = \theta - p_n$ into (3.33), we derive

$$\sum_{t_i>1}\left((p_n^2 - 6p_n + 2\theta p_n)t_i - 2p_n^2 + 2p_n - \theta p_n + 2\theta - 4\right) \leq 0.$$

Similarly, since $t_i \geq 2$ and $p_n^2 - 6p_n + 2\theta p_n > 0$, we have

$$\begin{aligned}
&(p_n^2 - 6p_n + 2\theta p_n)t_i - 2p_n^2 + 2p_n - \theta p_n + 2\theta - 4 \\
\geq &2p_n^2 - 12p_n + 4\theta p_n - 2p_n^2 + 2p_n - \theta p_n + 2\theta - 4 \\
= &3\theta p_n - 10p_n - 4 + 2\theta \geq 1.6 - 1.6p_n > 0.
\end{aligned}$$

So far, we have shown a contradiction to (3.31), and therefore, LB > UB.

Finally let us consider $\ell_1 > 0$. We first show that for $\ell_1 > 0$, the best case has the following properties:

1. $\ell_1 = \ell$ and $\ell_0 = N$;
2. The number of 1-machines, denoted by ℓ, is strictly more than $\sum_{t_i>1}(t_i - 1)$;

3. Denote by q_i^* and p_i^* the job assigned to 1-machine and $\tilde{1}$-machine in OPT respectively; have

$$q_1^* \geq \cdots \geq q_{\ell_0}^* \geq p_1^* \cdots \geq p_{\ell_1}^* \geq \theta - p_n > 2 - p_n \geq q_{\ell_0+1}^* \geq \cdots \geq q_{2\ell-\ell_0}^*;$$

4. The first largest $\sum_{t_i>1}(t_i - 1)$ jobs are assigned before LPT* assigns the first job to a 1-machine.

We would like to point out that Properties 2 and 4 are the same as Properties 1 and 3 in Section 3.4.3. While Property 3 is a special case of the previous Property 2 under the condition of $\ell_1 > 0$. We first show that if $\ell_1 > 0$, then a best case must satisfy $\ell_0 = N$.

Claim 3.6. *If M_i receives only one job in LPT*, then M_{i-1} must also receive only one job in LPT*.*

Proof. Denote by p_i and p_{i-1} the first job assigned to M_i and M_{i-1} respectively in LPT*. Accordingly we have $p_{i-1} \geq p_i$. Therefore, M_i will receive its second job before M_{i-1}. □

Since $\ell_1 > 0$, there exists a 1-machine M_i which receives only one job in LPT*. Together with Claim 3.6, we conclude that M_1 receives only one job in LPT*. Denote this job by p^*. According to (3.21), $W_1 = p^* \geq \theta - p_n$, which implies that jobs assigned before p^* are no smaller than $\theta - p_n > 1.8$. In a best case, these jobs are assigned to 1-machines in OPT; therefore, we have $\ell_0 \geq N$.

Denote $p_i \geq \theta - p_n$ $(i = 1, \ldots, \ell_1)$ the only job assigned to 1-machine M_i in LPT*. If p_i is assigned to a 1-machine $M_{i'}$ in OPT, then p_i must be the only job on $M_{i'}$ in OPT. This makes $W_{i'}^*$ and W_i cancel out each other in $\sum_{t_i=1}(W_i^* - W_i)$. Such p_i has no impact on increasing the upper bound of $\sum_{t_i=1}(W_i^* - W_i)$. As a result, we can assume that all the p_i are assigned to $\tilde{1}$-machines in OPT. Therefore $\ell_0 \leq N$, and together with $\ell_0 \geq N$, we derive $\ell_0 = N$.

Now apply LPT*. The largest ℓ_0 jobs, from q_1^* to $q_{\ell_0}^*$, are assigned to $\tilde{1}$-machines, before the next ℓ jobs being assigned to 1-machines. Among these ℓ jobs, the first ℓ_1 jobs, i.e., $p_1^*, \ldots, p_{\ell_1}^*$, are strictly greater than $\theta - p_n$. This is because they are the only jobs assigned to a 1-machine in LPT*. Additionally, as indicated by the best case, these jobs are assigned to $\tilde{1}$-machines in OPT. The remaining $\ell - \ell_1$ jobs, from $q_{\ell_0+1}^*$ to $q_{\ell_0+\ell-\ell_1}^*$ are assigned to 1-machines in OPT. Because these 1-machines

have two jobs in OPT, the sizes of q_i^* ($i = \ell_0 + 1, \ldots, \ell_0 + \ell - \ell_1$) are no bigger than $2 - p_n$. Table 3.11 illustrates the job allocation on 1-machines up to this point.

M_1	M_2	$\cdots$	M_{ℓ_1}	M_{ℓ_1+1}	M_{ℓ_1+2}	$\cdots$	M_ℓ
p_1^*	p_2^*	$\cdots$	$p_{\ell_1}^*$	$q_{\ell_0+1}^*$	$q_{\ell_0+2}^*$	$\cdots$	$q_{\ell_0+\ell-\ell_1}^*$
						$\cdots$	

Table 3.11: **Job allocation on 1-machines in LPT***

Among all the jobs that are assigned to 1-machines in OPT, there are still $\ell - 2\ell_0 + \ell_1$ left to be assigned. Assume that among these $\ell - 2\ell_0 + \ell_1$ jobs, N_1 are assigned to $\tilde{1}$-machines, and the other $\ell - 2\ell_0 + \ell_1 - N_1$ jobs, marked as $q_1, \ldots, q_{\ell-2\ell_0+\ell_1-N_1}$, are assigned to 1-machines. Finally, LPT* still needs to assign another $2\ell_0 - 2\ell_1 + N_1$ jobs to 1-machines and these jobs can only come from p_i^*. Mark these jobs as $p_1, \ldots, p_{2\ell_0-2\ell_1+N_1}$.

To sum up, the total jobs that are assigned to 1-machines in LPT* are: $p_1^*, \ldots, p_{\ell_1}^*$, $q_{\ell_0+1}^*, \ldots, q_{\ell_0+\ell-\ell_1}^*$, $q_1, \ldots, q_{\ell-2\ell_0+\ell_1-N_1}$, and $p_1, \ldots, p_{2\ell_0-2\ell_1+N_1}$. Therefore, we derive

$$\sum_{t_i=1}(W_i^* - W_i) = \sum_{i=1}^{\ell_0} q_i^* + \sum_{i=\ell_0+1}^{\ell_0+\ell-\ell_1} q_i^* + \sum_{i=\ell_0+\ell-\ell_1+1}^{2\ell-\ell_0} q_i^* - \left(\sum_{i=1}^{\ell_1} p_i^* + \sum_{i=\ell_0+1}^{\ell_0+\ell-\ell_1} q_i^* + \sum_{i=1}^{\ell-2\ell_0+\ell_1-N_1} q_i + \sum_{i=1}^{2\ell_0-2\ell_1+N_1} p_i \right).$$

If a job is assigned to a 1-machine in both OPT and LPT*, then it will be canceled out in the above equation. We apply the same technique by considering items that will not be canceled out, matching them into subsets, and finding the upper bound for each subset.

First allow $\ell_0 \geq \ell_1$, then $\ell_0 + \ell - \ell_1 > \ell$. For each $i \in \{1, \ldots, \ell_1\}$, $q_i^* < 2$ and $p_i^* > \theta - p_n$, and therefore,

$$q_i^* - p_i^* < 2 - \theta + p_n.$$

For each $i \in \{\ell_0+\ell-\ell_1+1, \ldots, 2\ell-\ell_0\}$, we can always find $i' \in \{\ell_0+1, \ldots, \ell-\ell_0+\ell_1\}$ such that q_i^* and $q_{i'}^*$ are assigned to the same 1-machine in OPT. Notice that $q_{i'}^*$ for $i' \in \{\ell_0 + 1, \ldots, \ell - \ell_0 + \ell_1\}$ is assigned to 1-machines in LPT*, and $q_{i'}^* + p_j \geq f$, where $f = \max\{2p_n, \theta - p_n\}$. If q_i^* is assigned to a 1-machine in LPT*, then it will

be canceled out in calculating the difference between $\sum_{t_i=1} W_i^*$ and $\sum_{t_i=1} W_i$. If q_i^* is assigned to a $\tilde{1}$-machine, then

$$(q_{i'}^* + q_i^*) - (q_{i'}^* + p_j) < 2 - f.$$

Finally for each $i \in \{\ell_1 + 1, \ldots, \ell_0\}$, and $j, j' \in \{1, \ldots, 2\ell_0 - 2\ell_1 + N_1\}$, we have

$$q_i^* - p_j - p_{j'} < 2 - 2p_n.$$

Accordingly,

$$\begin{aligned}\sum_{t_i=1}(W_i^* - W_i) <& \ell_1(2 - \theta + p_n) + (\ell_0 - \ell_1)(2 - 2p_n) + N_1(2 - f) \\ =& \ell_1(3p_n - \theta) + \ell_0(2 - 2p_n) + N_1(2 - f) = \text{UB}.\end{aligned}$$

Notice that $0 \le \ell_1 \le \ell_0$. If $p_n > \frac{\theta}{3}$, ℓ_1 should take its maximum possible value ℓ_0 in order to maximize UB; while if $p_n \le \frac{\theta}{3}$, we shall let ℓ_1 to take its minimum possible value 0 for the same purpose. The former case can be merged to the subsequent discussion for $\ell_1 \ge \ell_0$; the latter case, i.e., $\ell_1 = 0$, has already been considered. On the other hand, since $2 - 2p_n \ge 2 - f > 0$, we shall allow ℓ_0 to attain its biggest possible value, which is $\sum_{t_i>1}(t_i - 1)$.

Now assume that $\ell_1 \ge \ell_0$, then $\ell_0 + \ell - \ell_1 \le \ell$. for each $i \in \{1, \ldots, \ell_0\}$, $q_i^* < 2$ and $p_i^* > \theta - p_n$, and therefore,

$$q_i^* - p_i^* < 2 - \theta + p_n.$$

Among the N_1 jobs assigned to $\tilde{1}$-machines after p_1^*, assume that there are N_2 pairs of jobs being assigned to the same machine in OPT. Denote by q_i' and $q_{i'}'$ such a pair of jobs. Notice that $q_{\ell_0+1}^*, \ldots, q_{\ell_0+\ell-\ell_1}^*$ are assigned to 1-machines; therefore we have $N_2 \le \ell_1 - \ell_0$. For q_i' and $q_{i'}'$, we have

$$q_i' + q_{i'}' - p_j^* < 2 - \theta + p_n.$$

Among the remaining $N_1 - 2N_2$ jobs, take the smallest $2\ell_0 - 2\ell_1 + N_1$, denoted by $q_1'', \ldots, q_{2\ell_0-2\ell_1+N_1}''$. For each q_i'', we can find a corresponding job from $q_{i'}^*$ ($i' = \ell_0 + 1, \ldots, \ell_0 + \ell - \ell_1$), such that q_i'' and $q_{i'}^*$ are assigned to the same machine in

OPT. Therefore, we derive

$$(q^*_{i'} + q''_i) - (q^*_{i'} + p_j) < 2 - f.$$

For the remaining $2\ell_1 - 2\ell_0 - 2N_2$ jobs, denoted by q'''_i, we have

$$q'''_i + q'''_{i'} - p^*_j < 2 - \theta + p_n.$$

Accordingly,

$$\begin{aligned}\sum_{t_i=1}(W^*_i - W_i) <& \ell_0(2 - \theta + p_n) + N_2(2 - \theta + p_n) \\ &+ (2\ell_0 - 2\ell_1 + N_1)(2 - f) + (\ell_1 - \ell_0 - N_2)(2 - \theta + p_n) \\ =& \ell_0(4 - 2f) + \ell_1(p_n + 2f - \theta - 2) + N_1(2 - f) = \text{UB}.\end{aligned}$$

Because UB is independent to N_2, for simplicity, we let $N_2 = 0$. Since $2 - f > 0$, to derive the upper bound, we shall allow ℓ_0 and N_1 to attain their maximum values. Finally, we shall let $\ell_1 = 0$ when $p_n + 2f - \theta - 2 \leq 0$, and let ℓ_1 to attain its maximum value ℓ when $p_n + 2f - \theta - 2 > 0$. Notice that the former case has already been discussed.

In the best case, each 1-machine M_i receives exactly one job in LPT*, and as argued before, if this job is also assigned to some 1-machine $M_{i'}$ in OPT, then $W^*_{i'}$ and W_i will cancel out each other without impact on UB. Accordingly, the best case requires that $q^*_1 \geq \cdots \geq q^*_{\ell_0} \geq p^*_1 \geq \cdots \geq p^*_{\ell_1} \geq \theta - p_n$. In other words, all the jobs assigned to 1-machines in LPT* are assigned to $\tilde{1}$-machines in OPT.

For each $t_i \geq 2$, the number of 1-machines restricted by M_i is at most $\frac{2t_i}{\theta - p_n}$. The amount of workload can be compensated by each 1-machine is at most $2 - \theta + p_n$. Thus, the total workload compensation by 1-machines from M_i is at most

$$\text{UB}_i = \frac{2t_i}{\theta - p_n} \cdot (2 - \theta + p_n).$$

However, the minimum workload compensation required by M_i is

$$W_i - W^*_i > \theta t_i - p_n - 2t_i = \text{LB}_i.$$

Notice that $p_n + 2f - \theta - 2 > 0$ requires $p_n > \frac{2+\theta}{5} = 0.96$. Thus, for $t_i \geq 2$, $\theta > 2.8$

and $0.96 < p_n < 1$, we have

$$\begin{aligned}\mathrm{LB}_i - \mathrm{UB}_i =& \theta t_i - p_n - 2t_i - \frac{2t_i}{\theta - p_n} \cdot (2 - \theta + p_n) \\ =& \frac{(\theta^2 - p_n\theta - 4)t_i - p_n\theta + p_n^2}{\theta - p_n} \\ \geq& \frac{2\theta^2 - 3\theta p_n + p_n^2 - 8}{\theta - p_n} > 0.\end{aligned}$$

Apparently, the maximum workload can be compensated by the givers is not enough to cover the minimum workload required by the takers, i.e., LB > UB.

3.4.5 $\beta = 1$

When $\beta = 1$, both 1-machines and 2-machines can be givers. We have shown, for $\beta = 0$, the minimum workload required by the takers cannot be compensated by the maximum workload given by the givers in the counter-example I^*. For $\beta = 1$, we show that the total amount of workload which can be compensated by the givers will not increase as a result of the additional givers.

For $\beta = 1$ and $\theta \geq 2.8$, we derive $1.6 < p_n < 4$ from (3.22). If $p_n \geq 2$, then $W_i^* = 0$, $\forall\ M_i \in S_1$, due to $T_{\max}^* = 1$. Consequently, the 2-machines are the only givers when $p_n \geq 2$.

For $1.6 < p_n < 2$, assume that $p_n = 1.6 + \lambda$ $(0 < \lambda < 0.4)$. For $t_i = 1$, $W_i \geq p_n = 1.6 + \lambda$, and therefore

$$W_i^* - W_i < 2 - p_n = 0.4 - \lambda, \qquad \forall\ M_i \in S_1.$$

For $t_i = 2$, $W_i \geq 2\theta - p_n$, and therefore

$$W_i^* - W_i < 4 - 2\theta + p_n < \lambda, \qquad \forall\ M_i \in S_2.$$

The workload that can be compensated by a 1-machine is at most $0.4 - \lambda$, while the workload that can be compensated by a 2-machine is at most λ. Denote by ℓ_1 and ℓ_2 the total number of 1-machines and 2-machines. Thus

$$\sum_{t_i \in \{1,2\}} (W_i^* - W_i) < \ell_1(0.4 - \lambda) + \ell_2\lambda = \lambda(\ell_2 - \ell_1) + 0.4\ell_1 = \mathrm{UB}.$$

If $\ell_2 \geq \ell_1$, then UB is maximized when $\lambda \to 0.4$, however, this will lead to $W_i^* - W_i \to 0$ for all $M_i \in S_1$. In other words, 2-machines are the only givers. In addition, for $\lambda \to 0.4$, $p_n \to 2$, and $2p_n \to 4$, which implies that a giver 2-machine must receive one job only in LPT*. Thus, this situation can be merged with the case $p_n \geq 2$. Assume that all the 1-machines in instance I are deadwood, i.e., all the 1-machines are assigned with the same amount of workload in both OPT and LPT*. Then these 1-machines can be removed without influencing the workload assignment on the rest of the machines. Thus, instance I becomes exactly the same as $\beta = 0$, except that all the job sizes and machine speeds are doubled. We have shown that for such an instance, the total workload assigned in LPT* exceeds that in OPT. Now assume that some of the 1-machines in I are takers, i.e., they are assigned with more workload in LPT* than in OPT. With no additional givers compensating the extra workload, the total workload required in LPT* will only increase.

If $\ell_2 < \ell_1$, then UB is maximized when $\lambda \to 0$, and this will lead to $W_i^* - W_i \to 0$ for all $M_i \in S_2$. In other words, 1-machines are the only givers, and the instance becomes the same as for $\beta = 0$. As a result, it is safe to conclude that LB $>$ UB for $\beta = 1$.

We have shown the contradiction of LB $>$ UB for both $\beta = 0$ and $\beta = 1$ in the minimum counter example I^*. The contradiction implies that such I^* does not exist, and therefore we conclude $\theta < 2.8$.

3.5 Conclusion

The major contributions of this chapter are two-fold: Firstly, we show that any approximation algorithm for the makespan minimization problem can be directly applied to the deviation minimization problem, resulting in an approximation algorithm that can be as good as 2-approximate, depending on its performance on the original problem. Secondly, we provide an improved analysis on the worst-case performance of approximation algorithm LPT*. As a result, we reduce the current bound of LPT* from 3 to 2.8, which in combination with Theorem 3.3 leads to an efficient 2.8-approximation algorithm for the deviation minimization problem. The theorem further implies that when designing a truthful mechanism for the deviation minimization problem, one only needs to consider its performance from one direction (i.e., minimization of the makespan) as long as the approximation ratio is

above 2, even though the actual problem is concerned with both directions. The other direction (i.e., maximization of the cover) starts to play a binding role only after the algorithm has obtained a tight control over the makespan.

Chapter 4

Exact truthful mechanism for the deviation minimization problem

In the previous chapter we discussed the performance of some approximation algorithms on the deviation minimization objective function. In this chapter, we explore the nature of this new objective function with the aim of achieving a polynomial-time approximation scheme (PTAS).

4.1 Monotonicity of the deviation objective function

The makespan minimization and cover maximization problems can be both truthfully implemented with an exact algorithm. This property plays a key role in the design of mechanisms with good levels of performance (Archer and Tardos, 2001; Aulette et al., 2004). The success of the unified approach (Epstein et al., 2013), which provides PTASs for a wide class of uniform machine scheduling problems including the makespan minimization and the cover maximization problems, also relies on the monotone nature of the objective functions.

Although the new objective seems to be a simple combination of the makespan and the cover problems, it fails to inherit their monotone nature. This is because for the makespan and cover problems, the optimal values are measured from a fixed reference point of time 0. As a result, when changing the speed of an arbitrary machine, the optimal values will only change if the completion time of the changing machine reaches a point that deteriorates the current optimal value. By comparison,

the deviation function uses T_0 as a reference, the value of which is computed as the sum of all the job sizes divided by the sum of all the machine speeds. When speeds change, T_0 will also change. Consequently, the change of the optimal value may be caused purely by the change of T_0, rather than the machines that change their speeds.

Theorem 4.1. *The deviation minimization problem cannot be truthfully implemented with an exact algorithm.*

Proof. When there are multiple optimal solutions, an exact algorithm must specify a rule in order to output a unique solution. Accordingly, different exact algorithms may yield different results for the same instance. However, if there is only one optimal solution, then any exact algorithm will end up with the same solution. As a result, our proof is developed by showing that there exists a counter example with exactly one optimal solution both before and after reducing the speed of an arbitrary machine, and the workload on that machine under the optimal solution increases as a result of the reduction in speed.

Consider an instance of 5 machines and 9 jobs, with machine speeds $(13, 18, 21, 23, 26)$ sorted in non-decreasing order and jobs sizes $(122, 120, 87, 82, 77, 38, 21, 21, 20)$ sorted in non-increasing order. MATLAB is used to enumerate all $5^9 = 1,953,125$ feasible allocations, and among them a unique allocation outputs the optimal value for the deviation minimization objective. The job allocation and machine workload for this optimal solution are summarized below.

Allocation		Machines				
		$s_1 = 13$	$s_2 = 18$	$s_3 = 21$	$s_4 = 23$	$s_5 = 26$
Jobs	$p_1 = 122$			$\surd$		
	$p_2 = 120$					$\surd$
	$p_3 = 87$				$\surd$	
	$p_4 = 82$		$\surd$			
	$p_5 = 77$	$\surd$				
	$p_6 = 38$					$\surd$
	$p_7 = 21$				$\surd$	
	$p_8 = 21$				$\surd$	
	$p_9 = 20$		$\surd$			
Workload		77	102	122	129	158

Table 4.1: **Optimal allocation with the original speeds**

Next we reduce s_2 to 17.5 and resolve the maximum deviation of each feasible allocation based on the reduced speed. It turns out that the maximum deviation is

minimized with one unique allocation, with job assignment and workload shown as follows.

Allocation		Machines				
		$s_1 = 13$	$s_2 = 17.5$	$s_3 = 21$	$s_4 = 23$	$s_5 = 26$
Jobs	$p_1 = 122$			$\surd$		
	$p_2 = 120$				$\surd$	
	$p_3 = 87$					$\surd$
	$p_4 = 82$		$\surd$			
	$p_5 = 77$	$\surd$				
	$p_6 = 38$					$\surd$
	$p_7 = 21$		$\surd$			
	$p_8 = 21$					$\surd$
	$p_9 = 20$				$\surd$	
Workload		77	103	122	140	146

Table 4.2: **Optimal allocation with reduced speed of** M_2

For this specific instance, the optimal solution is unique both before and after reducing s_2. Consequently, any exact algorithm will yield the same result, i.e., M_2 receives a total workload of 102 when its reporting speed is 18, while it receives a total workload of 103 after its reporting speed reduces to 17.5. Apparently, the workload on M_2 increases due to a reduced reporting speed under any optimal algorithm. This completes our proof. □

In the above example, before speed reduction, the bottleneck machine in the optimal solution is M_5 – the one with the maximum completion time. After the speed reduction, if the optimal solution remains unchanged, then the bottleneck machine will become M_4 – the one with the minimum completion time. It is interesting to note that the completion times on both M_4 and M_5 remain the same. However the bottleneck machine changes from M_5 to M_4. This is because, after speed reduction, the average completion time T_0 increases and becomes closer to $T_{\max}$. If there exists another schedule which improves the optimal value, then the workload on M_4 in the new optimal solution must increase. The workload of M_2 – the machine that changes its speed, however, has no direct influence on the optimal value. Consequently, we are unable to decide whether it will increase or decrease. This is the underlying reason that prevents us from generating an exact truthful algorithm for the deviation minimization problem.

Although the non-existence of an exact truthful algorithm is conclusive, according

to our computational experiment, the counter example used in the proof only holds under some strict conditions. As a result, it is reasonable to check the conditions for such instances to exist.

4.2 Possible outcomes from reducing s_k

According to Archer and Tardos (2001, see Theorem 2.1), it is sufficient to prove a mechanism is truthful if we show that reducing the speed of an arbitrary machine, the workload assigned to it will not increase. In this section, we consider all possible changes of its workload when reducing the speed of an arbitary machine M_k. Define two sets of speeds S and S' as follows. Let M_k be an arbitrary chosen machine. For every $i \neq k$ the speed of M_i is s_i in both sets, and the speed of M_k is s_k and s'_k respectively, such that $s_k > s'_k$. Consider two instances I and I' with all else equal, except for the fact that the machine speeds in I are defined by S while the machine speeds in I' are defined by S'. Let Π^*_S and $\Pi^*_{S'}$ denote the sets of optimal solutions for I and I', respectively, under the deviation minimization objective. As s_k decreases to s'_k, the average completion time increases from T_0 to T'_0; consequently we define $T'_0 = T_0 + \delta$ for some $\delta > 0$. Let $\pi \in \Pi^*_S$ and $\pi' \in \Pi^*_{S'}$. Denote by Δ the optimal deviation for I, i.e., $\text{dev}^*(I) = \text{dev}(\pi, S) = \Delta$. Notice that π is still a feasible (not necessarily optimal) solution for I', and so is π' for I; accordingly there are four combinations of allocations and speed sets:

- Allocation π under the original speed set S, denoted by (π, S);
- Allocation π under the reduced speed set S', denoted by (π, S');
- Allocation π' under the original speed set S, denoted by (π', S);
- Allocation π' under the reduced speed set S', denoted by (π', S').

We subsequently examine various possibilities when reducing the speed of an arbitrary machine.

Lemma 4.2. *Arbitrarily choose π from Π^*_S. If dev$(\pi, S') = T_{\max}(\pi, S') - T'_0$, then $W_k(\pi) \geq W_k(\pi')$ for all $\pi' \in \Pi^*_{S'}$, unless $\pi \in \Pi^*_{S'}$ and $T_{\max}(\pi, S) = T_{\max}(\pi, S') = T_0 + \Delta$.*

Proof. Since $\Pi^*_{S'}$ represents the set of optimal solutions for instance I', for all $\pi' \in \Pi^*_{S'}$, we have $\text{dev}(\pi', S') \leq \text{dev}(\pi, S')$. Together with $\text{dev}(\pi', S') \geq T_{\max}(\pi', S') - T'_0$ and $\text{dev}(\pi, S') = T_{\max}(\pi, S') - T'_0$, this implies that

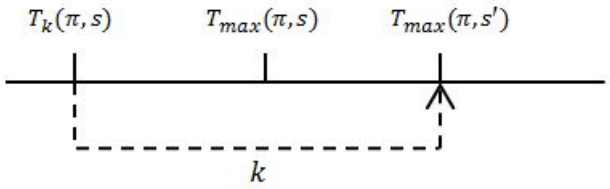
$$T_{\max}(\pi', S') \leq T_{\max}(\pi, S'). \tag{4.1}$$

In addition, since only M_k reduces its speed from s_k to s'_k, $T_{\max}(\pi, S) \leq T_{\max}(\pi, S')$ must hold.

1. If $T_{\max}(\pi, S') > T_{\max}(\pi, S)$, then

$$T_{\max}(\pi, S') = T_k(\pi, S').$$

Together with (4.1), we derive $T_k(\pi, S') = T_{\max}(\pi, S') \geq T_{\max}(\pi', S') \geq T_k(\pi', S')$. Thus, $W_k(\pi) \geq W_k(\pi')$ is derived directly from $T_k(\pi, S') \geq T_k(\pi', S')$.



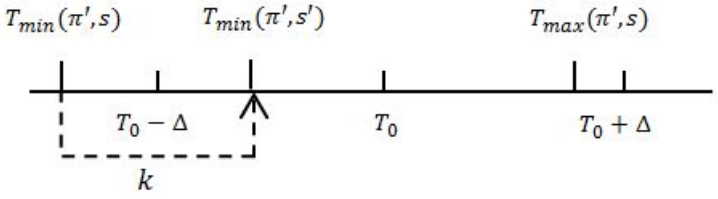
2. Subsequently, we assume that $T_{\max}(\pi, S') = T_{\max}(\pi, S)$.

   (a) If $T_{\max}(\pi, S') = T_{\max}(\pi, S) < T_0 + \Delta$, then together with (4.1), we have

$$T_{\max}(\pi', S) \leq T_{\max}(\pi', S') \leq T_{\max}(\pi, S') < T_0 + \Delta. \tag{4.2}$$

In order for $\pi \in \Pi^*_S$, it requires $\text{dev}(\pi', S) \geq \text{dev}(\pi, S) = \Delta$, which together with (4.2) implies the following:

$$T_{\min}(\pi', S) \leq T_0 - \Delta. \tag{4.3}$$



However, since $T'_0 > T_0$ and $T_{\max}(\pi, S') = T_{\max}(\pi, S)$, we derive $\text{dev}(\pi, S) \geq T_{\max}(\pi, S) - T_0 > T_{\max}(\pi, S') - T'_0 = \text{dev}(\pi, S')$. This implies that

$$T_{\min}(\pi', S') > T_0 - \Delta, \tag{4.4}$$

because $T_0' - T_{\min}(\pi', S') \leq \text{dev}(\pi', S') \leq \text{dev}(\pi, S') = T_{\max}(\pi, S') - T_0' < T_0 + \Delta - T_0' = \Delta - \delta$. The combination of (4.3) and (4.4) suggests that

$$T_{\min}(\pi', S') > T_{\min}(\pi', S).$$

Therefore, it can only be the case that $T_k(\pi', S) = T_{\min}(\pi', S)$. Together with (4.3), we derive $T_k(\pi', S) \leq T_0 - \Delta \leq T_{\min}(\pi, S) \leq T_k(\pi, S)$; thus $W_k(\pi) \geq W_k(\pi')$ is again straightforward.

(b) If $T_{\max}(\pi, S') = T_{\max}(\pi, S) = T_0 + \Delta$, then we only consider $\pi \notin \Pi^*_{S'}$ since $\pi \in \Pi^*_{S'}$ is excluded from our statement. If $\pi \notin \Pi^*_{S'}$, then $\text{dev}(\pi', S') < \text{dev}(\pi, S')$, which implies that

$$T_{\max}(\pi', S) \leq T_{\max}(\pi', S') < T_{\max}(\pi, S') = T_{\max}(\pi, S).$$

However, due to $\pi \in \Pi^*_S$, from $\text{dev}(\pi', S) \geq \text{dev}(\pi, S) = \Delta$ and $T_{\max}(\pi', S) < T_0 + \Delta$, we derive

$$T_{\min}(\pi', S) \leq T_0 - \Delta.$$

On the other hand, the fact that $\text{dev}(\pi', S') < \text{dev}(\pi, S') = T_{\max}(\pi, S') - T_0' = \Delta - \delta$ requires

$$T_{\min}(\pi', S') > T_0' - (\Delta - \delta) = T_0 - \Delta + 2\delta > T_{\min}(\pi', S).$$

Accordingly, $T_k(\pi', S) = T_{\min}(\pi', S) \leq T_0 - \Delta \leq T_k(\pi, S)$, from which we derive $W_k(\pi) \geq W_k(\pi')$.

□

According to Lemma 4.2, if an optimal solution π is chosen from Π^*_S such that after reducing the speed of an arbitrary machine M_k, $T_{\max}(\pi, S')$ becomes the bottleneck, then one can choose any allocation from $\Pi^*_{S'}$ and the result is monotone. The only exception is when $T_{\max}(\pi, S) = T_{\max}(\pi, S') = T_0 + \Delta$ and $\pi \in \Pi^*_{S'}$. Notice that an exceptional case requests $\pi \in \Pi^*_{S'}$. Thus, if an exact algorithm uniquely chooses π from both Π^*_S and $\Pi^*_{S'}$, then we can achieve at least $W_k(\pi) = W_k(\pi')$. The result is still monotone.

Next, we consider the case where the bottleneck for (π, S') is $T_{\min}(\pi, S')$.

Lemma 4.3. *Arbitrarily choose π from Π^*_S. If dev$(\pi, S') = T_0' - T_{\min}(\pi, S')$ and $\pi \notin \Pi^*_{S'}$, then $\forall\ \pi' \in \Pi^*_{S'}$, either $W_k(\pi) \geq W_k(\pi')$ or $T_0 + \Delta \leq T_{\max}(\pi', S') <$*

$T_0 + \Delta + 2\delta$.

Proof. We first assume that $\mathrm{dev}(\pi', S') < \Delta - \delta$, from which we derive $T_{max}(\pi', S') \leq T'_0 + \mathrm{dev}(\pi', S') < T'_0 + \Delta - \delta = T_0 + \Delta$, which in turn implies that

$$T_{\max}(\pi', S) < T_0 + \Delta. \tag{4.5}$$

Since $\mathrm{dev}(\pi', S) \geq \mathrm{dev}(\pi, S) = \Delta$, together with (4.5), we derive $T_{\min}(\pi', S) \leq T_0 - \Delta$. However, due to $\pi \notin \Pi^*_{S'}$, we require $\mathrm{dev}(\pi, S') > \mathrm{dev}(\pi', S')$ for all $\pi' \in \Pi^*_{S'}$. Together with $\mathrm{dev}(\pi, S') = T'_0 - T_{\min}(\pi, S')$ and $\mathrm{dev}(\pi', S') \geq T'_0 - T_{\min}(\pi', S')$, we derive

$$T_{\min}(\pi', S') > T_{\min}(\pi, S') \geq T_{\min}(\pi, S) \geq T_0 - \Delta.$$

Since only M_k reduces its speed in set S', it is easy to derive $T_k(\pi', S) = T_{\min}(\pi', S) < T_{\min}(\pi, S) \leq T_k(\pi, S)$, and therefore, $W_k(\pi) \geq W_k(\pi')$.

Next we assume that $\mathrm{dev}(\pi', S') \geq \Delta - \delta$. Then, at least one of the following inequalities is true for $\pi \notin \Pi^*_{S'}$:

$$T_{\max}(\pi, S') > T'_0 + \mathrm{dev}(\pi', S') \geq T_{\max}(\pi', S'); \tag{4.6}$$

$$T_{\min}(\pi, S') < T'_0 - \mathrm{dev}(\pi', S') \leq T_{\min}(\pi', S'). \tag{4.7}$$

- If (4.6) is true, then $T_{\max}(\pi, S') > T'_0 + \mathrm{dev}(\pi', S') \geq T'_0 + \Delta - \delta = T_0 + \Delta \geq T_{\max}(\pi, S)$, which implies that $T_k(\pi, S') = T_{\max}(\pi, S')$. Thus $T_k(\pi, S') = T_{\max}(\pi, S') \geq T_{\max}(\pi', S') \geq T_k(\pi', S')$, and therefore, $W_k(\pi) \geq W_k(\pi')$.
- If (4.7) is true, then first assume that $T_{\max}(\pi', S') < T_0 + \Delta$, which implies (4.5), and $W_k(\pi) \geq W_k(\pi')$ can be derived with the same logic as for $\mathrm{dev}(\pi', S') < \Delta - \delta$. Otherwise, we have $T_{\max}(\pi', S') \geq T_0 + \Delta$. Finally, since $\mathrm{dev}(\pi, S') = T'_0 - T_{\min}(\pi, S') \leq T'_0 - T_0 + \Delta = \Delta + \delta$, we derive $\mathrm{dev}(\pi', S') < \mathrm{dev}(\pi, S') \leq \Delta + \delta$, and therefore, $T_{\max}(\pi', S') \leq T'_0 + \mathrm{dev}(\pi', S') < T_0 + \Delta + 2\delta$. This completes our proof.

□

For an arbitrary π, either $\mathrm{dev}(\pi, S') = T_{\max}(\pi, S') - T'_0$ or $\mathrm{dev}(\pi, S') = T'_0 - T_{\min}(\pi, S')$. Therefore, the combination of Lemma 4.2 and 4.3 implies that monotonicity can be achieved with any exact algorithm in the deviation minimization problem unless an instance has the following properties.

Definition 4.1. *For a pair of instances I and I', if the outputs from any exact*

algorithm A satisfy $W_k^A(I) \geq W_k^A(I')$, then we say truthfulness is achievable for free for instance I.

Proposition 4.4. *Truthfulness is not free to achieve for I under the deviation minimization objective if I has one of the following properties:*

Property 1. $\pi \in \Pi^*_{S'}$, and either (a) or (b) holds true:

(a) $dev(\pi, S') = T'_0 - T_{\min}(\pi, S')$;

(b) $dev(\pi, S') = T_{\max}(\pi, S') - T'_0$ and $T_{\max}(\pi, S) = T_{\max}(\pi, S') = T_0 + \Delta$.

Property 2. $\pi \notin \Pi^*_{S'}$ and $T_0 + \Delta \leq T_{\max}(\pi', S') < T_0 + \Delta + 2\delta$.

4.3 An exact algorithm with improved performance on truthfulness

Theorem 4.1 summarizes a negative result of designing an exact algorithm that is always truthfully implementable. However, we have also shown that for many instances, monotonicity can be achieved for free. We propose an algorithm that guarantees truthfulness for a high proportion among instances where truthfulness is not achievable for free.

4.3.1 An exact truthful algorithm for the makespan problem

In this section, we explain how an exact truthful algorithm works for the makespan problems and why it fails in the case of our new objective function.

Consider a makespan minimization problem with a set of speeds S. Assume without loss of generality that there are ℓ optimal solutions, denoted by $\Pi = \{\pi_1, \ldots, \pi_\ell\}$, and for arbitrary M_k, we have $W_k(\pi_1) \geq W_k(\pi_2) \geq \cdots \geq W_k(\pi_\ell)$. Partition Π into ℓ' subsets in such a way that the schedules in the same subset have the same workload on M_k. In mathematical terms, $\Pi = \bigcup_{i=1}^{\ell'} \Pi_i$, where $\Pi_i = \{\pi_i | W_k(\pi_i) = c_i\}$ and $c_1 > c_2 > \cdots > c_{\ell'}$.

Reduce the speed of M_k by ε while keeping the speeds of other machines unchanged. Starting with $\varepsilon = 0$, we gradually increase the value of ε. For any feasible schedule, the only feature that will change during this process is the completion

time of M_k. The entire process can be split into two periods that appear alternatively. In the first period, the optimal solution set is Π as long as $\forall\ \pi_i \in \Pi_1$, $T_k(\pi_i) = W_k(\pi_i)/(s_k - \varepsilon) \leq T_{\max}$. The optimal solution set will become $\Pi \setminus \Pi_1$ if ε keeps increasing to the point that $W_k(\pi_j)/(s_k - \varepsilon) \leq T_{\max} < W_k(\pi_i)/(s_k - \varepsilon)$, $\forall\ i \in \Pi_1$ and $\forall\ j \in \Pi_2$. Similarly, as we keep increasing ε, the optimal solution set will become $\Pi \setminus (\Pi_1 \cup \Pi_2)$, $\Pi \setminus (\Pi_1 \cup \Pi_2 \cup \Pi_3)$, and so on, until the optimal solution set becomes $\Pi_{\ell'}$. In the second period, speed s_k reduces to the extend that the optimal solution set becomes a complete different set Π', where $\Pi' \cap \Pi = \emptyset$.

A few exact algorithms can be truthfully implemented for the makespan minimization problem. One which is widely used in the literature sorts the workload vector of each optimal solution in a non-decreasing order, and chooses the one whose workload vector is the lexicographically minimum among all the optimal solutions (Archer and Tardos, 2001). For simplicity, we name this algorithm Lexicographically Minimum (LM).

In the first period, each time a subset is excluded from the optimal set, it is the one that assigns the greatest workload to M_k. Thus, as long as an algorithm selects a unique solution based on the same standard which is independent of the machine speeds, the algorithm is truthfully implementable.

As for the second period, truthfulness can be achieved for free. This is because each solution in Π' assigns at most $c_{\ell'}$ amount of workload to M_k. The success of LM relies on the fact that the only factor that deteriorates the optimal value is given by the completion time of M_k. As discussed previously, this property is not found in the deviation minimization objective.

4.3.2 A new selection rule

The exceptional instances summarized in Proposition 4.4 fall into two types. The first type only contains instances for which no exact algorithm can yield a truthful outcome. Some instances with Property 2, like the one given at the beginning of this chapter, fall into this group. The second type contains those instances for which at least one exact algorithm can provide a truthful result. Some of the instances with Property 2 and all the instances with Property 1 fall into this group. For such instances, applying LM may lead to monotonicity. However, the performance of LM is completely random. In this section, we aim to find an algorithm which guarantees truthfulness for most, if not all, instances that belong to the second type.

Claim 4.1. *Any set of feasible allocations* Π *can be divided into two mutually exclusive and collectively exhaustive subsets* $\Pi^{\min}$ *and* $\Pi^{\max}$*, where*

- $\Pi^{\min} = \{\pi | T_{\max} + T_{\min} < 2T_0\}$*;*
- $\Pi^{\max} = \{\pi | T_{\max} + T_{\min} \geq 2T_0\}$.

Tie breaking rule (TBR)

Let Π^* be the entire set of optimal solutions for instance I and $|\Pi^*| > 1$. A tie among the optimal solutions is broken by the following steps.

Step 1. Classify the elements in Π^* so that $\Pi^* = \Pi^{\min} \cup \Pi^{\max}$.

Step 2. If $\Pi^{\min} \neq \emptyset$, sort all the elements in $\Pi^{\min}$ in non-increasing order based on the completion time of each machine. Choose the one that is lexicographically minimum as the final solution.

Step 3. Otherwise, sort all the elements in $\Pi^{\max}$ in non-decreasing order based on the completion time of each machine. Choose the one that is lexicographically maximum as the final solution.

Lemma 4.5. *TBR is a truthful implementation for the deviation minimization problem if* $\pi \in \Pi^*_{S'}$*, unless* $T_{\max}(\pi, S) = T_{\max}(\pi', S) = T_{\max}(\pi, S') = T_{\max}(\pi', S') = T_0 + \Delta$ *and* $T_{\min}(\pi, S) = T_{\min}(\pi', S) = T_{\min}(\pi, S') = T_{\min}(\pi', S')$.

Proof. Classify the elements in Π^*_S and $\Pi^*_{S'}$, respectively, so that $\Pi^*_S = \Pi^{\min}_S \cup \Pi^{\max}_S$ and $\Pi^*_{S'} = \Pi^{\min}_{S'} \cup \Pi^{\max}_{S'}$. Let $\pi \in \Pi^*_S$ and $\pi' \in \Pi^*_{S'}$ be the solution chosen by TBR for I and I' respectively. We prove that $W_k(\pi) \geq W_k(\pi')$.

According to Proposition 4.4, for $\pi \in \Pi^*_{S'}$, monotonicity is achieved with any $\pi \in \Pi^*_S$, except when (a) is true or (b) is true. Therefore, we only need to consider the two exceptional cases. Notice that if $\pi = \pi'$, then $W_k(\pi) = W_k(\pi')$; hence we make the assumption of $\pi \neq \pi'$.

1. If $\text{dev}(\pi, S') = T'_0 - T_{\min}(\pi, S')$, then according to $\pi \in \Pi^*_{S'}$, we derive $\pi \in \Pi^{\min}_{S'}$, i.e., $\Pi^{\min}_{S'} \neq \emptyset$, which in turn implies that $\pi' \in \Pi^{\min}_{S'}$. Sort π and π' in non-increasing order respectively, based on completion times computed under speeds S'. As TBR chooses π' over π, π' is lexicographically smaller than π, and therefore

$$T_{\max}(\pi, S') \geq T_{\max}(\pi', S'). \tag{4.8}$$

In addition, $\text{dev}(\pi, S') = \text{dev}(\pi', S') = T'_0 - T_{\min}(\pi', S')$ implies that

$$T_{\min}(\pi', S') = T_{\min}(\pi, S'). \tag{4.9}$$

(a) If $\pi' \notin \Pi^*_S$, then at least one of the following inequalities is true:

$$T_{\max}(\pi', S) > T_0 + \Delta; \tag{4.10}$$

$$T_{\min}(\pi', S) < T_0 - \Delta. \tag{4.11}$$

- First let (4.10) be true. Together with (4.8), we derive

$$T_{\max}(\pi, S') \geq T_{\max}(\pi', S') \geq T_{\max}(\pi', S) > T_0 + \Delta \geq T_{\max}(\pi, S),$$

  which in turn implies that $T_k(\pi, S') = T_{\max}(\pi, S') \geq T_{\max}(\pi', S') \geq T_k(\pi', S')$; thus, $W_k(\pi) \geq W_k(\pi')$ is straightforward.
- Next let (4.11) be true. Together with (4.9), we derive

$$T_{\min}(\pi', S') = T_{\min}(\pi, S') \geq T_{\min}(\pi, S) \geq T_0 - \Delta > T_{\min}(\pi', S),$$

  which in turn implies that $T_k(\pi', S) = T_{\min}(\pi', S)$. Thus $W_k(\pi) \geq W_k(\pi')$ is straightforward since $T_k(\pi', S) < T_0 - \Delta \leq T_k(\pi, S)$.

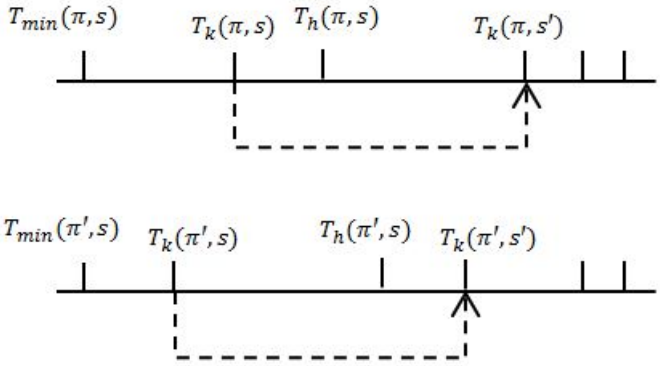
(b) If $\pi' \in \Pi^*_S$, then either $\pi' \in \Pi^{\min}_S$ or $\pi' \in \Pi^{\max}_S$.

i. First assume that $\pi' \in \Pi^{\min}_S$, which implies that $\Pi^{\min}_S \neq \emptyset$, and therefore $\pi \in \Pi^{\min}_S$. Sort π and π' respectively in non-increasing order based on completion times calculated under speeds S. As TBR chooses π over π', π is lexicographically smaller than π'. In other words, there exists h such that $T_h(\pi, S) < T_h(\pi', S)$ and $T_i(\pi, S) = T_i(\pi', S)$, $\forall\ i < h$. However, π' is lexicographically smaller than π under speeds S'. Since only M_k reduces its speed from s_k to s'_k, this is only possible if $T_k(\pi, S') > T_h(\pi', S') > T_h(\pi, S')$ and $T_k(\pi', S') < T_k(\pi, S')$. The latter implies that $W_k(\pi) \geq W_k(\pi')$.

ii. Next assume that $\pi' \in \Pi^{\max}_S$. Accordingly, $T_{\max}(\pi', S) = T_0 + \Delta$.

If $\pi \in \Pi^{\min}_S$, then $T_{\max}(\pi, S) < T_0 + \Delta$. However, according to (4.8), $T_{\max}(\pi, S') \geq T_{\max}(\pi', S') \geq T_0 + \Delta$. This implies that $T_{\max}(\pi, S') = T_k(\pi, S') \geq T_{\max}(\pi', S') \geq T_k(\pi', S')$, and therefore $W_k(\pi) \geq W_k(\pi')$.

If $\pi \in \Pi^{\max}_S$, then $T_{\max}(\pi, S) = T_{\max}(\pi', S) = T_0 + \Delta$. Sorting π



and π' in non-decreasing order based on completion times calculated under speeds S, π is lexicographically bigger than π'. In other words, $T_{\min}(\pi', S) \leq T_{\min}(\pi, S)$.

- If $T_{\min}(\pi', S) < T_{\min}(\pi, S)$, together with (4.9), we derive

$$T_{\min}(\pi', S) < T_{\min}(\pi, S) \leq T_{\min}(\pi, S') = T_{\min}(\pi', S'),$$

  which in turn implies $T_{\min}(\pi', S) = T_k(\pi', S) < T_{\min}(\pi, S) \leq T_k(\pi, S)$, and therefore $W_k(\pi) \geq W_k(\pi')$.
- For $T_{\min}(\pi', S) = T_{\min}(\pi, S)$, if

$$T_{\min}(\pi', S) = T_{\min}(\pi, S) < T_{\min}(\pi', S') = T_{\min}(\pi, S'),$$

  then $T_{\min}(\pi', S) = T_k(\pi', S)$ and $T_{\min}(\pi, S) = T_k(\pi, S)$. In other words, $W_k(\pi) = W_k(\pi')$.
- Finally, if $T_{\min}(\pi', S) = T_{\min}(\pi, S) = T_{\min}(\pi', S') = T_{\min}(\pi, S')$, since $T_{\max}(\pi, S) = T_{\max}(\pi', S) = T_0 + \Delta$, according to our statement, we only consider $T_{\max}(\pi, S') > T_0 + \Delta$, $T_{\max}(\pi', S') > T_0 + \Delta$, or both. On the other hand, the fact that TBR picks π' over π in $\Pi^{\min}_{S'}$ implies that $T_{\max}(\pi', S') \leq T_{\max}(\pi, S')$. Therefore, it must be the case that $T_{\max}(\pi, S') \geq T_{\max}(\pi', S') \geq T_{\max}(\pi, S)$. Therefore, $T_{\max}(\pi, S') = T_k(\pi, S') \geq T_k(\pi', S') \implies W_k(\pi) \geq W_k(\pi')$.

2. If $\text{dev}(\pi, S') = T_{\max}(\pi, S') - T'_0$ and $T_{\max}(\pi, S) = T_{\max}(\pi, S') = T_0 + \Delta$, then $\text{dev}(\pi, S) = T_{\max}(\pi, S) - T_0$, i.e., $\pi \in \Pi^{\max}_S$, which implies that $\Pi^{\min}_S = \emptyset$. Additionally, since $\text{dev}(\pi', S') \leq \text{dev}(\pi, S') = T_{\max}(\pi, S') - T'_0 = \Delta - \delta$, π'

satisfies

$$T_{\min}(\pi', S') \geq T'_0 - (\Delta - \delta) = T_0 - \Delta + 2\delta. \tag{4.12}$$

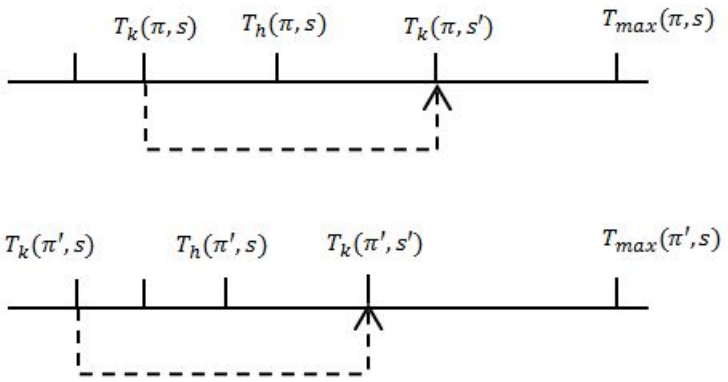
(a) If $\Pi^{\min}_{S'} \neq \emptyset$, then

$$T_{\max}(\pi', S) \leq T_{\max}(\pi', S') < T_{max}(\pi, S') = T_0 + \Delta,$$

due to $\pi \in \Pi^{\max}_{S'}$ and $\pi' \in \Pi^{\min}_{S'}$. However, $\text{dev}(\pi', S) \geq \text{dev}(\pi, S) = \Delta$, therefore π' must satisfy $T_{\min}(\pi', S) \leq T_0 - \Delta$. Together with (4.12), we derive $T_{\min}(\pi', S') > T_{\min}(\pi', S)$, which implies that $T_{\min}(\pi', S) = T_k(\pi', S) < T_0 - \Delta \leq T_k(\pi, S)$; thus $W_k(\pi) \geq W_k(\pi')$ is straightforward.

(b) If $\Pi^{\min}_{S'} = \emptyset$, then both π and π' belong to $\Pi^{\max}_{S'}$. Sorting π and π' in non-decreasing order based on completion times calculated under speeds S', π' is lexicographically bigger than π.

- If $\pi' \notin \Pi^*_S$, then it must be the case that $T_{\min}(\pi', S) < T_0 - \Delta$, since $T_{\max}(\pi', S) \leq T_0 + \Delta$. Again, with (4.12), it implies that $T_{\min}(\pi', S) = T_k(\pi', S) < T_0 - \Delta \leq T_k(\pi, S)$ and therefore $W_k(\pi) \geq W_k(\pi')$.
- If $\pi' \in \Pi^*_S$, then both π and π' are elements of $\Pi^{\max}_S$. We sort π and π' respectively in non-decreasing order based on completion times calculated under speeds S, and π is lexicographically bigger than π'. In other words, there exists h such that $T_h(\pi, S) > T_h(\pi', S)$ and $T_l(\pi, S) = T_l(\pi', S)$, $\forall\ l < h$. Nevertheless, π' is lexicographically bigger than π in $\Pi^{\max}_{S'}$. Accordingly, it must be the case that $T_k(\pi', S') < T_h(\pi, S)$ and $T_k(\pi', S) < T_k(\pi, S)$. The latter implies that $W_k(\pi) \geq W_k(\pi')$.



□

As an immediate result of Lemma 4.5, we derive the following statement regarding the truthfulness of TBR.

Proposition 4.6. *TBR is truthfully implementable for the deviation minimization problem, unless an instance satisfies one of the following conditions.*

1. $\pi \in \Pi^*_{S'}$, $T_{\max}(\pi, S) = T_{\max}(\pi', S) = T_{\max}(\pi, S') = T_{\max}(\pi', S') = T_0 + \Delta$ *and* $T_{\min}(\pi, S) = T_{\min}(\pi', S) = T_{\min}(\pi, S') = T_{\min}(\pi', S')$.
2. $\pi \notin \Pi^*_{S'}$ *and* $T_0 + \Delta < T_{\max}(\pi', S') < T_0 + \Delta + 2\delta$.

It is worth noticing that both conditions can be easily violated if the reporting speeds increases/decreases by a relatively big amount. More specifically, rules can be made so that each time M_k changes its speed, the amount it is allowed to change must be no less than ε. If ε is too big, then the accuracy of the output will be sacrificed. However, if ε is too small, then the above conditions may not be efficiently avoided. To ensure TBR is truthfully implementable for any instances, a lower bound of ε can be found by solving $\frac{W_k(\pi)}{s_k - \varepsilon} \geq T_0 + \Delta + 2\delta$. However, in this inequality, δ is a function of ε, and s_k is some private information unavailable to the central authority. This increases the difficulties in generating a concrete function for ε.

4.3.3 Combine TBR with the unified approach

The unified approach proposed by Epstein et al. (2013) can be modified to solve the deviation minimization problem. The result is PTAS but not monotone.

Proposition 4.7. *Assume that* $s_1 \leq s_2 \leq \cdots \leq s_m$. *There exists an optimal schedule* π *for the deviation minimization problem which satisfies* $W_1(\pi) \leq W_2(\pi) \leq \cdots \leq W_m(\pi)$.

Proof. Consider a schedule π with maximum deviation Δ. A pair of machines M_i and M_j is called reversed if $1 \leq i < j \leq m$ and $W_i(\pi) > W_j(\pi)$. We show that removing a consecutive reversed pair (that is, $j = i + 1$) by swapping the workloads assigned to them from any schedule π does not increase the maximum deviation. Let π' be the schedule resulting from swapping the two job sets of M_i and M_j. In π', the completion time of M_i is bounded by

$$T_0 - \Delta \leq \frac{W_j(\pi)}{s_j} \leq \frac{W_i(\pi')}{s_i} = \frac{W_j(\pi)}{s_i} < \frac{W_i(\pi)}{s_i} \leq T_0 + \Delta,$$

while the completion time of M_j is bounded by

$$T_0 - \Delta \leq \frac{W_j(\pi)}{s_j} \leq \frac{W_j(\pi')}{s_j} = \frac{W_i(\pi)}{s_j} \leq \frac{W_i(\pi)}{s_i} \leq T_0 + \Delta.$$

□

Proposition 4.7 presents a desirable property which enables us to apply the unified approach to the deviation minimization problem. In their work *A unified approach to truthful scheduling on related machines*, Epstein et al. (2013) define a highly structured schedule which can be found by constructing a directed graph via dynamic programming. Moreover, they present the following theorem.

Theorem 4.8 (Epstein et al., 2013)**.** *Given a schedule π such that $W_1(\pi) \leq W_2(\pi) \leq \cdots \leq W_m(\pi)$, there exists a structured schedule π^*, such that for $i = 1, 2, \ldots, m$, we have*

$$(1 - 14\varepsilon) \cdot W_i(\pi) \leq W_i(\pi^*) \leq (1 + 14\varepsilon) \cdot W_i(\pi). \tag{4.13}$$

According to Proposition 4.7. for any deviation minimization problem I, there exists an optimal solution π, such that $W_1(\pi) \leq W_2(\pi) \leq \cdots \leq W_m(\pi)$. Thus, we can always find the corresponding structured schedule of π, denoted by π^*, via generating the directed graph under the unified approach as if I was a makespan problem. Since π is an optimal solution, we have $\text{dev}^*(I) = \max\{T_{\max}(\pi) - T_0, T_0 - T_{\min}(\pi)\}$. Moreover,

$$(1 - 14\varepsilon) \cdot T_i(\pi) \leq T_i(\pi^*) \leq (1 + 14\varepsilon) \cdot T_i(\pi), \ \forall\ i$$

is derived from (4.13) based on Theorem 4.8. Therefore, we have

$$\text{dev}(\pi^*) \leq (1 + 14\varepsilon) \cdot \text{dev}^*(I).$$

The structured schedule π^* can be found in the following way. First, generate the directed layered graph under the unified approach. Then, replace the weights of each vertex with dev_i, where i presents the corresponding machine in layer i. Each optimal structured schedule corresponds to a path that minimizes the maximum weight of a vertex along the path. When more than one structured schedule can be found, the tie is broken by TBR.

The resulting algorithm is a PTAS with runtime $\mathcal{O}(n^{((14r+19)\lambda+8)})$, where $r \geq 5$ and $\lambda = \lceil \log_{1+\varepsilon} 2 \rceil$, and monotonicity is determined by TBR.

4.4 Computational experiment

To gauge the ability of TBR in providing good solutions in terms of monotonicity, a computational experiment was conducted. The algorithms were coded in MATLAB and implemented on a server with 3.4 GHz dual processor, 8 GB RAM memory, and Windows 10 operating system. Due to the purpose of the experiment, the complete enumeration of m^n needs to be considered. On the one hand, to test the performance of TBR, we are interested in the instances with more than one optimal solution. If the number of jobs is close to or smaller than the number of machines, then more often than not, such instances present only one optimal solution. This is the case for instances with $m = 6$ and $n = 8$. On the other hand, to test for monotonicity, we prefer instances with larger numbers of machines. This is because monotonicity can often be achieved for free if an instance has three or even fewer machines. As presented in Table 4.3, the largest instances that we can generate the complete enumeration of without breaking the memory are $m = 5$, $n = 9$ and $m = 4$, $n = 11$. Computational results from the two combinations show little difference. As a result, we only present our results from running the experiments for $m = 5$ and $n = 9$.

m \ n	8	9	10	11	12
4	5.126s	10.562s	41.347s	174.088s	out of memory
5	13.602s	67.656s	out of memory		
6	53.839s	out of memory			

Table 4.3: **Impact of instance sizes on memory**

Based on Proposition 4.7, the problem can be further sized down by only considering the structured schedules. This efficiently reduces the number of schedules from $5^9 = 1,953,125$ to around 10^4 for $m = 5$ and $n = 9$, and the resulting computation time is reduced by around 30 times (from approx. 340s to 11s of CPU time). Uniform distributions are employed to generate the job sizes and machine speeds. Varying the ranges of these two variables shows little impact on the final results. In each run, a random number between 1 and 5 is given to k, the index of the machine that changes its speed. The amount of speed being reduced follows a continuous uniform distribution between 0 and 4. Table 4.4 summarizes the main variables in

MATLAB, and the way they are generated. Both code and data are available on demand.

Variables	Explanation	Property	Range
job_sizes	The size of each job.	Discrete	$[1, 100]$
machine_speeds	The speed of each machine.	Discrete	$[1, 50]$
k	The index of machine that changes its speed.	Discrete	$[1, 5]$
reduce_amount	The amount of speed being reduced.	Continuous	$(0, 4]$

Table 4.4: **Data generation summary**

The computational experiment consists of two parts. The first part compares the performance of TBR and LM, in particular when monotonicity is not free to achieve. The second part demonstrates the performance of TBR with rounding speeds.

4.4.1 TBR vs. LM

This section discusses the results of applying TBR to various instances and compares them with the results of applying LM to the same instances. In order to measure performance, we introduce notation n^f, n^T and n^L to count the number of instances in which (a) monotonicity is achieved for free, (b) monotonicity is achieved under TBR, and (c) monotonicity is achieved under LM. We first initialize the values of n^f, n^T and n^L to zero. For each of the randomly generated instances, we obtain the optimal solution set for the original speeds and the reduced speeds, respectively. If monotonicity can be achieved effortless, that is $\min_{\pi \in \Pi^*_S}\{W_k(\pi)\} \geq \max_{\pi' \in \Pi^*_{S'}}\{W_k(\pi')\}$, then

$$n^f := n^f + 1;\ n^T := n^T + 1;\ n^L := n^L + 1.$$

Otherwise, we apply TBR and LM respectively. If the result of applying TBR is monotone, i.e., $W_k^{TBR}(\pi) \geq W_k^{TBR}(\pi')$, then $n^T := n^T + 1$. Likewise, if the result of applying LM is monotone, i.e., $W_k^{LM}(\pi) \geq W_k^{LM}(\pi')$, then $n^L := n^L + 1$.

A total number of 5000 randomly generated instances were tested, and we derive $n^f = 2854$, $n^T = 4993$, and $n^L = 4944$. Not surprisingly, TBR provides an improved performance (4993 out of 5000 instances, or 99.86%) in comparison to that of LM (4944 out of 5000 instances, or 98.88%). Given that in more than half of the instances ($n^f = 2854$), monotonicity can be achieved with any exact algorithm, it

is more appropriate to compare the following probabilities:

$$\frac{n^T - n^f}{5000 - n^f} = 99.67\%; \ \frac{n^L - n^f}{5000 - n^f} = 97.39\%.$$

The new results indicate that when truthfulness is not achievable for free, TBR provides truthful results with a probability of 99.67%, comparing to 97.39% by LM. The improvement is much less significant than we expected due to an underestimation of the performance of LM. Both candidates provide nearly 100% positive results. However, as long as there exists a non-monotone instance, the algorithm is not truthful. The computational results coincide with our theoretical results.

4.4.2 TBR with rounding speeds

This section is designed to test the performance of TBR when it is used in combination with rounding speeds. The original machine speeds and the reduced speeds in the previous 5000 instances are both rounded down to the largest integer powers of 2. The optimal solution sets are then derived based on the rounded speeds. We apply TBR and LM respectively when monotonicity is not for free, count the number of the three types of instances in the same way as described in the first experiment, and denote them by n_r^f, n_r^T and n_r^L, respectively to distinguish from the previous results.

Denote by π^* and π respectively the optimal solution under the deviation minimization objective with the original speeds and the rounding speeds. Due to the use of rounding speeds, the objective value under solution π is no longer global optima. We employ

$$a_1 = \frac{\text{dev}(\pi) + T_0}{\text{dev}(\pi^*) + T_0}$$

to measure the approximation ratio of TBR under rounding speeds.

Another guaranteed truthful algorithm is the exact algorithm for the makespan problem with tie broken by LM. Based on Theorem 3.3, the alternative algorithm provides 2-approximation for the deviation minimization problem. For each instance, we obtain its optimal solution under the makespan minimization objective with the original speeds, and denote the solution by π'. We use

$$a_2 = \frac{\text{dev}(\pi') + T_0}{\text{dev}(\pi^*) + T_0}$$

to measure the approximation ratio of the alternative algorithm. If TBR yields 100% positive results when running with rounding speeds, we would like to further compare the approximation ratios a_1 and a_2.

Figure 4.1 compares the histograms of a_1 and a_2. Both graphs are heavily skewed to the left. However, the histogram of a_2 has a long tail and a small peak after 1.5. By comparison, the distribution of a_1 mainly centers between 1 and 1.5. From the histograms, we can conclude that by rounding the speeds to the powers of 2, we are able to eliminate the exceptional instances where TBR fails to be monotone. Although the output allocation is no longer optimal, it is a more accurate approximation to the original problem than simply solving it as a makespan problem. A closer examination of individual instances also suggests that the current speed rounding rule is quite relax. We conjecture that TBR can be converted into a truthful algorithm under a tighter speed rounding rule.

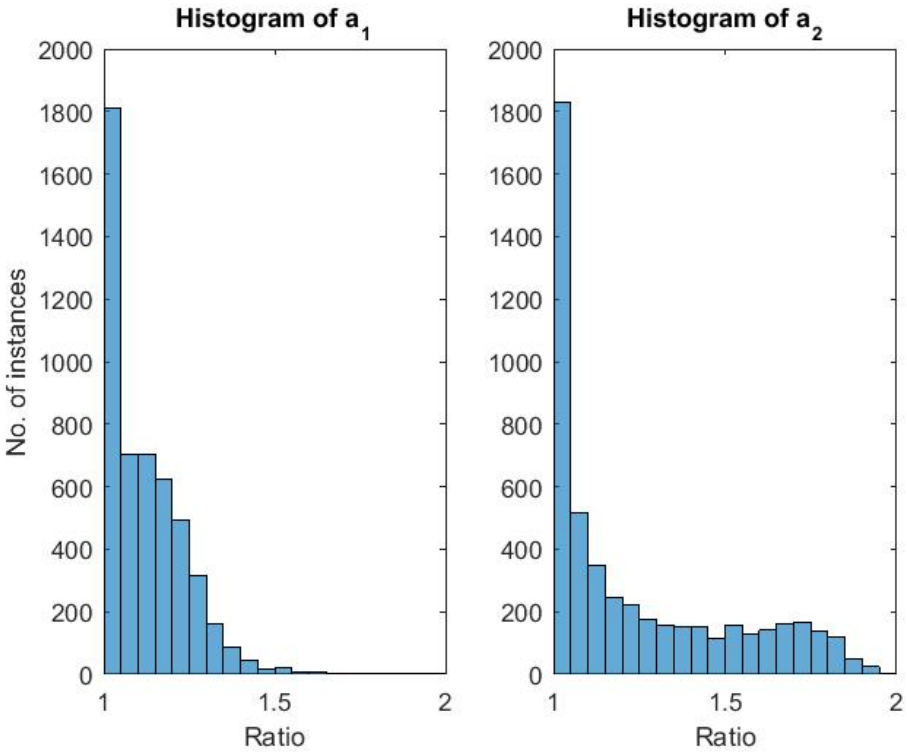


Figure 4.1: Histograms

4.5 Conclusion

If the new objective function inherits the monotone nature from the makespan and the cover problems, then to design a truthful PTAS will be straightforward. Unfortunately, no exact algorithm can be truthfully implementable to the deviation minimization problem. Having said that, there are only a few cases where when an agent reports a reduced/increased speed, its workload may increase/decrease instead. These cases have been characterized precisely in Proposition 4.4. To deal with the exceptional instances, we propose TBR, which imposes much more strict conditions for instances where truthfulness is not guaranteed. Based on our further observation, those conditions no longer hold if minor changes in reporting speeds are not allowed. Computational results also suggest a high potential for converting TBR into a truthful algorithm with good approximation ratio when rounding speeds are employed.

Chapter 5

Tardiness minimization problem

In the previous chapters, fairness has been defined from the machines' point of view. This chapter, however, takes an alternative approach, and considers the issue of fairness from the perspective of the jobs.

5.1 Background and motivation

Our model is inspired by observations from the home health care scheme, a service that aims to provide clinical care that is usually available in hospitals with the only difference that in this case it takes place at a patient's home. In other words, rather than having patients travel to the hospital, practitioners are sent to patients' houses. As the service is provided outside the hospital with little control imposed, practitioners may misreport the time they spend with a patient. On the other hand, each patient receives an ideal cure time, and the aim of the hospital is to deliver the service before this due date. Due to the limitations on the available resources, not all patients' requirements can be fulfilled in time. One of the objectives is therefore to minimize the maximum delay among all the patients. To this end, it is essential for the hospital to solicit the correct information from practitioners.

The above objective is also known as the maximum tardiness minimization problem. One of the earliest studies of this type can be dated back to the 1950s. Jackson (1955) considered the one-machine scheduling problem with the objective of minimizing maximum lateness, a feature that is measured by the difference between job completion time and job due date. The notion of tardiness, defined as the maximum value between 0 and lateness, was originally proposed by Emmons (1969). It was argued that tardiness is often a more reasonable substitute to lateness. Garey and

Johnson (1979) proved that the maximum tardiness problem is NP-hard even with a single machine. Tardiness related objectives have been considered under various settings. Examples are given by: Potts and Van Wassenhove (1985), Alidaee and Rosa (1997), and Biskup et al. (2008), who consider the total (weighted) tardiness problem; Kayvanfar et al. (2014), and Balakrishnan et al. (1999), who combine tardiness with earliness under the influence of just-in-time philosophy; and Yang et al. (2004) and Kim et al. (2003), who consider tardiness under overtime costs and setup times, respectively. Results on tardiness feature with single machine are rich, although many of them are enumerative based (Baker, 1974; Hall et al., 1991). A fair amount of research have been conducted for identical machines (Azizoglu and Kirca, 1998) as well as for unrelated parallel machines (Kayvanfar et al., 2014). However, the literature becomes quite scant when we combine the tardiness feature with uniform machines. The study that seems to be the closest to this work was carried out by Koulamas and Kyparisis (2000), who extend the earliest due date (EDD) rule to scheduling problems with uniform machines in order to minimize maximum lateness. The result only exceeds the optimal value by no more than $p_{\max}$, where $p_{\max}$ represents the maximum job length. To the best of our knowledge, the maximum tardiness minimization objective has not been considered in the field of mechanism designs with machine agents.

5.1.1 Model description

In the uniform machine scheduling model, we assume that each job J_j is associated to a due date d_j by which it is ideally completed. Let $c_j(\pi)$ denote the completion time of J_j under a feasible schedule π. The lateness of J_j is defined by $L_j(\pi) = c_j(\pi) - d_j$, and $\text{tard}_j(\pi) = \max\{0, c_j(\pi) - d_j\}$ defines its tardiness. The objective is then to find a schedule π that minimizes the maximum tardiness among all the jobs. In mathematical terms,

$$\min_{\pi} \max_{j} \text{tard}_j(\pi).$$

As our modifications do not change the game theoretical nature of the problem, for this model the condition for a truthful algorithm follows Archer and Tardos (2001).

No assumptions have been made between a job's due date and its size. Therefore, these two features are treated as independent of each other. For an algorithm to be truthful, its output workloads must be monotone with regard to the machine speeds; while in order to optimize the tardiness objective, one only takes into account the due dates. Intuitively, a truthful algorithm will perform poorly on the

tardiness objective.

Denote by tard^* the optimal objective value of a maximum tardiness minimization problem, and tard^A the objective value derived from applying algorithm A. The performance of A can be measured by

$$\phi^A = \frac{\text{tard}^A}{\text{tard}^*}.$$

However, tard^* may equal zero for some instances. To avoid a zero denominator, we adjust ϕ^A by adding a constant $d_{\max} = \max_j\{d_j\}$ to both the top and the bottom as suggested by Lenstra (1976). The resulting approximation ratio becomes

$$\phi^A = \frac{\text{tard}^A + d_{\max}}{\text{tard}^* + d_{\max}}.$$

5.1.2 Some general results

This section presents the results related to the monotone nature of the tardiness objective and the link with the makespan problem. Since p_j is independent of d_j, the following conclusion is straightforward.

Proposition 5.1. *The maximum tardiness minimization problem cannot be truthfully implemented with an exact algorithm.*

Proof. According to Theorem 2.1, to prove this statement, we only need to show the existence of an instance which has only one optimal solution both before and after reducing the speed of an arbitrary machine, and the workload of that machine increases due to its speed reduction.

	M_1	M_2	c_j	tard_j
J_1	$\surd$		0.5	0
J_2	$\surd$		1.8	0.4
J_3		$\surd$	2	0.4

Table 5.1: **Optimal solution with original speeds**

Consider an instance with two machines of speeds $s_1 = 10, s_2 = 7$. We assign three jobs of sizes $p_1 = 5, p_2 = 13, p_3 = 14$, and the corresponding due dates $d_1 = 1, d_2 = 1.4, d_3 = 1.6$ to the machines in such a way that the maximum tardiness among the jobs is minimized. In the optimal solution, the two jobs with smaller sizes are assigned to M_1 and the biggest job is assigned to M_2. The optimal value

is 0.4, as shown in Table 5.1.

We reduce the speed of M_1 to 9 while all the other values remain the same. The optimal solution for the new instance, presented in Table 5.2, is to assign J_1 and J_3 to M_1 and J_2 to M_2. This gives us a maximum tardiness of $\frac{23}{45}$.

	M_1	M_2	c_j	tard_j
J_1	$\surd$		$\frac{5}{9}$	0
J_2		$\surd$	$\frac{13}{7}$	$\frac{16}{35}$
J_3	$\surd$		$2\frac{1}{9}$	$\frac{23}{45}$

Table 5.2: **Optimal solution with reduced speeds**

As the speed of the faster machine reduces from 10 to 9, the work assigned to it increases from 18 to 19. □

As in the case of the deviation minimization problem, the maximum tardiness minimization objective is not truthful in nature. However, Proposition 5.2 enables us to apply any existing truthful algorithm for the makespan problem to the tardiness problem and the performance of the algorithm only deteriorates by 1.

Proposition 5.2. *If algorithm A is an α-approximation algorithm for the makespan minimization objective, then A is $\alpha + 1$-approximate for the maximum tardiness minimization problem.*

Proof. Let π^A denote the output from the application of algorithm A, while π^T and π^M respectively define the schedule that optimizes the maximum tardiness, and the one that optimizes the maximum makespan. Let $T_{\max}$ and $T^*_{\max}$ denote the makespan of π^A, and π^M respectively. Since A is α-approximate for the makespan problem, we have that $T_{\max} \leq \alpha T^*_{\max}$. Finally, M_k and $M_{k'}$ respectively define the bottleneck machines in π^A and π^T. Thus,

$$\phi^A = \frac{\max\{d_{\max}, c_k(\pi^A) - d_k + d_{\max}\}}{\max\{d_{\max}, c_{k'}(\pi^T) - d_{k'} + d_{\max}\}}.$$

First, we assume that $d_{\max} \geq T^*_{\max}$, then

$$\begin{aligned}\phi^A &= \frac{\max\{d_{\max}, c_k(\pi^A) - d_k + d_{\max}\}}{\max\{d_{\max}, c_{k'}(\pi^T) - d_{k'} + d_{\max}\}} \\ &\leq \frac{c_k(\pi^A) - d_k + d_{\max}}{d_{\max}} \leq \frac{\alpha T^*_{\max} + d_{\max}}{d_{\max}} \\ &\leq \frac{\alpha d_{\max} + d_{\max}}{d_{\max}} = \alpha + 1.\end{aligned}$$

The second inequality is due to $c_k(\pi^A) \leq T_{\max} \leq \alpha T^*_{\max}$ and $d_k > 0$.

Next, we assume that $d_{\max} < T^*_{\max}$, then

$$c_{k'}(\pi^T) - d_{k'} + d_{\max} \geq T^*_{\max} - d_{\max} + d_{\max} = T^*_{\max} > d_{\max},$$

from which we derive

$$\begin{aligned}\phi^A &= \frac{\max\{d_{\max}, c_k(\pi^A) - d_k + d_{\max}\}}{\max\{d_{\max}, c_{k'}(\pi^T) - d_{k'} + d_{\max}\}} \\ &\leq \frac{c_k(\pi^A) - d_k + d_{\max}}{c_{k'}(\pi^T) - d_{k'} + d_{\max}} \leq \frac{\alpha T^*_{\max} + d_{\max}}{T^*_{\max}} \\ &< \frac{\alpha T^*_{\max} + T^*_{\max}}{T^*_{\max}} = \alpha + 1.\end{aligned}$$

□

5.2 Two-stage LPT*

In this section, we aim to develop a heuristic for the tardiness minimization problem that serves the following objectives: (a) speediness in allocating jobs (b) truthfulness in soliciting private information concerning the different speeds and (c) efficiency in achieving fairness among the jobs. Let us first consider a special condition relating to job lengths and due dates. Specifically, it is assumed that for J_j and $J_{j'}$, if $p_j > p_{j'}$ then $d_j \geq d_{j'}$. Under this assumption, LPT becomes equivalent to EDD, which sorts jobs according to non-decreasing due dates, and assigns the next available job in the sequence to the earliest finishing machine. It is known that the adjusted LPT is a truthful algorithm for uniform machine scheduling problem, whereas EDD is an efficient greedy algorithm for tardiness related objectives. Indeed, the performance of LPT on the maximum tardiness minimization problem largely depends on the due date range, i.e., $d_{\max} - d_{\min}$. As long as this range is relatively small, the job

sequence in LPT will not differ too much from the job sequence in EDD. Based on this observation, we propose a two-stage LPT* with the aim of buffering the impact from a large due date range.

Lemma 5.3. *Two-stage LPT* is a truthful algorithm.*

Proof. In sub case i, the workload received by each machine is equal to the sum of the workloads computed by a monotone algorithm (LPT*) running on two disjoint sets of jobs. These two sets are uniquely determined by the case index i, and are independent of the machine speeds. □

Two-stage LPT*

Input: a job sequence σ, and a speed vector S.

Step 1. Sort the job sequence so that $d_1 \leq d_2 \leq \cdots \leq d_n$.

Step 2. In sub case i ($i = 1$), break σ into two partial sequences σ_1 and σ_2 so that $\sigma_1 = \{J_1, \ldots, J_i\}$, and $\sigma_2 = \{J_{i+1}, \ldots, J_n\}$.

Step 3. Run LPT* independently on σ_1 and σ_2. The final jobs on M_k are the jobs assigned to M_k from both sequences.

Step 4. For each M_k, re-sort the jobs assigned to it according to non-decreasing due dates.

Step 5. Compute the maximum tardiness based on the re-ordered job positions and record the result as $\text{tard}(i)$.

Step 6. $i := i + 1$. Go to Step 2 until $i := n$.

Step 7. Output $\text{tard}^{\text{TL}^*} = \min\{\text{tard}(i)\}$ and the corresponding job positions as the final results.

Sub case n in two-stage LPT* outputs the same result as LPT*. In other words, $\text{tard}^{\text{TL}^*} \leq \text{tard}(n)$. This implies that two-stage LPT* performs at least as well as LPT*. In Section 3.4 we have shown that LPT* is a 2.8-approximation algorithm for the makespan minimization problem. Hence, according to Proposition 5.2, LPT* is at most a 3.8-approximation for the maximum tardiness minimization problem.

5.3 Computational tests

This section is designed to evaluate the performance of two-stage LPT*. The algorithm is coded in MATLAB. The number of machines (m), and the number of jobs (n) are set to be $\{3, 5, 10\}$ and $\{10, 20, 50, 100\}$ respectively. Both the job sizes and the machine speeds are generated from a discrete uniform distribution between 1 and 100. The due dates follow $U[(T_{\max}(1-\alpha-\beta/2), T_{\max}(1-\alpha+\beta/2)]$, a distribution suggested by Potts and Van Wassenhove (1985). Parameters α and β control the average tardiness and the due date range. We fix α to be 0.5, and consider both $\beta = 0.2$ and $\beta = 0.8$. Since makespan $T_{\max}$ cannot be calculated directly, the average completion time T_0 is used as an approximation. A batch of 100 random instances are created for each type of combination.

The Earliest Due Date (EDD) and Shortest Processing Time (SPT) are among the earliest results for tardiness related objectives. Both algorithms undertake the same procedures as LPT in job allocation, except that EDD orders jobs according to non-decreasing due dates, and SPT orders jobs according to non-decreasing job sizes. The two algorithms still play a fundamental role in designing many of the heuristic procedures. A common belief is that a highly congested system indicates the use of SPT, whereas EDD performs well when congestion is low. Since the optimal solution for the maximum tardiness minimization problem is NP-complete, we compare the results of two-stage LPT* with those obtained by implementing EDD and SPT instead. The result of two-stage LPT* is also compared with the result of LPT* to see how much improvement in the approximation ratio is obtained at the expense of additional computational costs.

We used the criteria suggested by Alidaee and Rosa (1997) to compare the behaviours of the different algorithms.

(1) Percentage of Number of Best (PNoB): This refers to the percentage of the number of times in which an algorithm provides a schedule which is at least as good as the others.

(2) Average of the Ratio (Ratio): This refers to the average of the following ratio.

$$Ratio = \frac{\text{tard}^A - \text{tard}^*}{\text{tard}^*} \times 100,$$

where tard^* denotes the best results achieved by implementing all the algorithms that we would like to compare.

(3) Maximum of Ratio (MAR): This is the maximum of the above Ratio.

(4) CPU: This is the total CPU time of running 100 random instances.

The main results are presented in Table 5.3. Our purpose is not to compare two-stage LPT* with EDD or SPT, as the latter two algorithms are not truthful. Instead, we aim to examine whether the improvement of two-stage LPT* in approximation ratio is significant enough to justify its additional computations compared with LPT*. Having said that, it is worthwhile noticing that with a small number of jobs (within 3 times the number of machines), two-stage LPT* outperforms all three other rules. Additionally, with small size problems ($n \leq 20$), the computation time in running two-stage LPT* is negligible.

Performances of the algorithms in tight and loose due date range settings based on the problem sizes are visualized in Figures 5.1-5.3. With the number of jobs increasing by multiples of the number of machines, the advantages of two-stage LPT* over LPT* become less obvious when the due date range is tight. Nevertheless, two-stage LPT* manages to maintain its level of performance under a loose due date range, while the performance of LPT* shows a clear deterioration as the due date range increases.

As indicated by MAR, two-stage LPT* is very efficient in controlling bad cases in comparison to LPT*. This advantage is highlighted in the case of loose due date ranges. This is a clear evidence of an improved approximation ratio by applying two-stage LPT* to the maximum deviation minimization problem. Finally, although computation times increase significantly after $n \geq 50$, the absolute running time for a single instance is quite reasonable (max 9.74s CPU time for $m = 10$, $n = 100$).

5.4 Conclusion

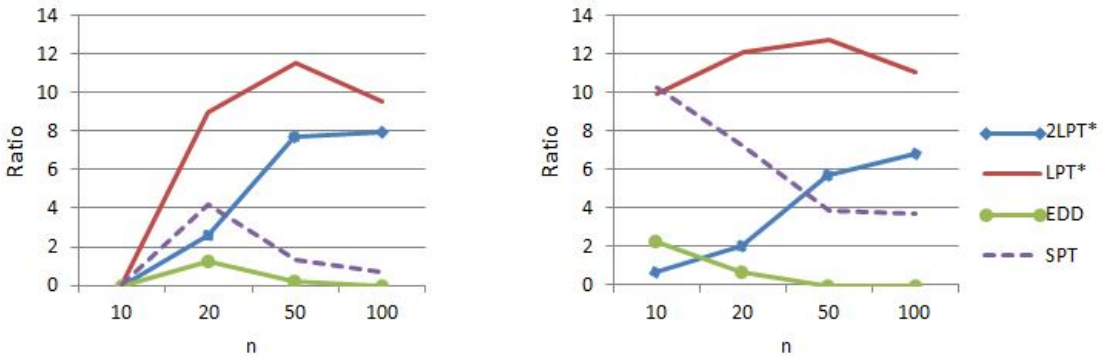
As an extension of the notion of fairness, this chapter addressed the issue of fairness from the perspective of the jobs. An application of this problem can be found in home health care schemes. Due to the lack of link between the size of a job and the due dates, the objective function is not monotone in nature. However, one is still able to achieve a good approximation ratio with existing truthful algorithms for the makespan problem. In particular, LPT* presents a 3.8-approximation for the maximum tardiness minimization problem. To develop a more accurate algorithm for the target objective, efforts were directed to combining the monotonicity feature

of LPT* with EDD, a traditional algorithm which is designed to address tardiness related problems. The resulting two-stage LPT* remains monotone and presents much improved performance with regard to the computational results, particularly in the case of small size problems and loose due date ranges.

n	β	**PNoB**				**Ratio**				**MAR**				**CPU**	
		2LPT*	LPT*	EDD	SPT	2LPT*	LPT*	EDD	SPT	2LPT*	LPT*	EDD	SPT	2LPT*	LPT*
$m = 3$															
10	0.2	70%	12%	24%	7%	1.02	6.10	4.23	8.07	9.86	30.68	18.88	24.47	0.91	0.34
	0.4	68%	4%	33%	4%	0.71	9.90	2.24	10.26	6.37	36.31	14.06	37.54	1.00	0.33
20	0.2	42%	0%	45%	13%	2.64	8.97	1.27	4.19	16.05	42.10	7.16	14.12	3.88	0.48
	0.4	39%	0%	59%	3%	1.99	12.12	0.66	7.22	12.34	41.83	4.17	27.49	3.98	0.47
50	0.2	6%	0%	74%	20%	7.73	11.55	0.22	1.36	28.56	42.66	2.50	9.95	42.16	1.17
	0.4	8%	0%	88%	4%	5.66	12.70	0.07	3.84	27.38	44.91	1.29	16.77	41.99	1.20
100	0.2	6%	0%	79%	15%	7.92	9.58	0.07	0.72	42.11	47.47	0.99	2.20	302.07	3.47
	0.4	2%	0%	97%	1%	6.77	11.06	0.01	3.65	25.12	55.37	0.72	15.36	292.63	3.49
$m = 5$															
10	0.2	90%	31%	12%	0%	0.41	4.53	7.78	14.27	7.60	17.89	30.01	30.57	1.24	0.41
	0.4	72%	8%	26%	4%	0.85	9.84	4.36	12.47	12.48	41.60	23.72	38.04	1.31	0.40
20	0.2	62%	1%	33%	5%	1.34	9.17	3.07	8.00	10.72	27.97	12.13	24.06	5.69	0.62
	0.4	43%	1%	56%	2%	1.46	12.69	1.12	9.86	10.58	52.48	7.69	35.07	5.81	0.61
50	0.2	6%	0%	88%	6%	7.44	13.31	0.16	2.67	35.41	44.91	3.57	10.19	65.12	1.78
	0.4	5%	0%	94%	1%	5.70	15.52	0.05	8.71	16.03	54.36	2.97	24.64	64.94	1.68
100	0.2	2%	0%	85%	13%	10.05	13.40	0.06	1.50	24.96	28.78	1.13	6.69	477.27	5.59
	0.4	0%	0%	100%	0%	9.91	16.52	0	6.41	23.09	48.06	0	29.59	469.88	5.48
$m = 10$															
10	0.2	97%	86%	5%	0%	0.01	0.55	16.06	25.84	0.61	17.28	47.80	60.58	1.91	0.49
	0.4	86%	52%	18%	1%	0.30	2.83	6.31	17.84	6.07	16.75	24.78	44.14	2.06	0.49
20	0.2	97%	20%	3%	0%	0.16	5.52	9.59	17.17	6.95	21.94	25.68	36.49	10.18	0.87
	0.4	83%	0%	18%	0%	0.24	13.03	3.92	18.18	4.91	33.77	12.75	44.08	10.14	0.83
50	0.2	33%	1%	61%	6%	2.92	10.85	0.92	5.27	12.00	30.53	5.68	13.40	123.39	3.10
	0.4	2%	0%	98%	0%	4.94	19.79	0.02	11.67	12.15	43.90	1.15	33.56	124.69	3.11
100	0.2	0%	0%	87%	13%	9.43	15.87	0.10	2.42	27.97	37.32	1.95	10.07	915.40	10.48
	0.4	0%	0%	87%	13%	9.38	20.93	0	10.57	18.54	48.14	0	34.06	920.20	10.64

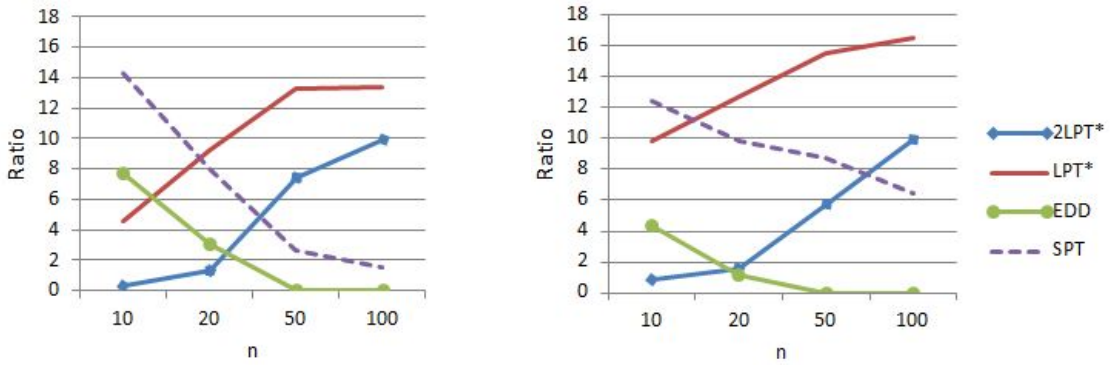
Note: 2LPT* refers to Two-stage LPT*

Table 5.3: **Computational results**



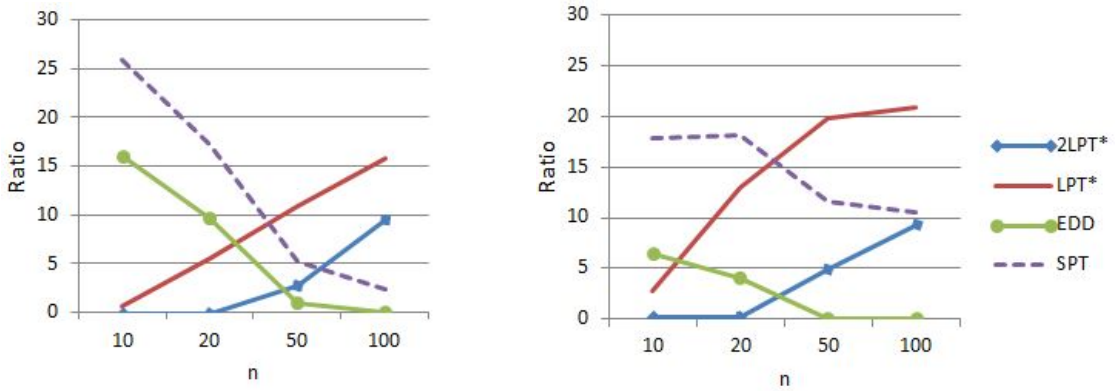
(a) Tight due date range $\beta = 0.2$ (b) Loose due date range $\beta = 0.4$

Figure 5.1: **Ratio for different algorithms** $m = 3$



(a) Tight due date range $\beta = 0.2$ (b) Loose due date range $\beta = 0.4$

Figure 5.2: **Ratio for different algorithms** $m = 5$



(a) Tight due date range $\beta = 0.2$ (b) Loose due date range $\beta = 0.4$

Figure 5.3: **Ratio for different algorithms** $m = 10$

Chapter 6

Summary and concluding remarks

We have proposed a new objective function that better reflects the concept of fairness in the uniform machine scheduling setting. In the new objective, named deviation minimization function, the completion times of machines are bounded from both directions so that a more even distribution of jobs to machines can be achieved. Unfortunately, lack of monotonicity in the nature of the objective jeopardized our attempt to develop a truthful PTAS for the problem. On the other hand, the close link between the deviation minimization and the makespan minimization problems enables us to employ truthful algorithms designed for the latter directly to our targeted problem and maintain good approximation ratios. In particular, we have looked into a fast, previously known as 3-approximation algorithm – LPT*. Using the approach of a minimum counter-example, we have established an improved upper bound of 2.8, and it is believed that, with the same approach, a tight bound of the algorithm can be found.

Based on the fact that the optimal value under the deviation minimization function can be bounded by either the maximum or the minimum completion time, we have developed a tie breaking rule (TBR) which selects a unique optimal solution based on the bottleneck machine. The algorithm only fails the monotonicity test on instances satisfying strict conditions. In addition, those conditions can be easily violated, and thus the failure instances can be eliminated, if the central authority imposes rounding rule on the speeds revealed by the machine agents. For instance, our computational results suggest that TBR passes the monotonicity test with zero failure when agents are requested to report their speeds in the form of 2 to the

power. In fact, we conjecture that a much tighter rounding rule could serve the same purpose, and it is clear that the tighter the rounding rule is, the smaller the approximation ratio will become.

As an extension to the deviation minimization problem, we have also considered the maximum tardiness minimization, a traditional objective that meets the needs for fairness among jobs with due dates, under the game theoretical settings with uniform machines. To tackle this problem, we have proposed two-stage LPT* which efficiently balances the conflicts between the demand for monotone workload from truthfulness against the demand for minimizing maximum tardiness from the objective function.

Our work raises the problem of designing truthful PTASs when the objective function itself is not monotone. To the best of our knowledge, studies on mechanism design in machine scheduling have been limited to objective functions that are monotone in nature, i.e., the exact algorithm for the objective function is truthfully implementable. A PTAS for these problems is usually generated by reducing the amount of enumeration to be considered under the exact algorithm. However, the existing approaches fail in the situations when the objective function is no longer monotone.

Another area that is worth further study is the application of different fairness criteria in machine scheduling. In our work, fairness is considered in the max-min form, and is limited to a single dimension, i.e., the completion time for either machines or jobs. But the notion is usually much more complex in reality. Therefore, it is worthwhile to consider the issue of fairness based on other theories or to bring the it into real contexts.

Bibliography

Afek, Y., Mansour, Y. and Ostfeld, Z., 1996, July. Convergence complexity of optimistic rate based flow control algorithms. *In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (pp. 89-98). ACM.

Alidaee, B. and Rosa, D., 1997. Scheduling parallel machines to minimize total weighted and unweighted tardiness. *Computers & Operations Research*, 24(8), pp.775-788.

Andelman, N., Azar, Y. and Sorani, M., 2005, February. Truthful approximation mechanisms for scheduling selfish related machines. *In Annual Symposium on Theoretical Aspects of Computer Science* (pp. 69-82). Springer Berlin Heidelberg.

Annamalai, C., Kalaitzis, C. and Svensson, O., 2017. Combinatorial algorithm for restricted max-min fair allocation. *ACM Transactions on Algorithms* (TALG), 13(3), p.37.

Archer, A. and Tardos, É., 2001, October. Truthful mechanisms for one-parameter agents. *In Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on* (pp. 482-491). IEEE.

Asadpour, A., Feige, U. and Saberi, A., 2008. Santa claus meets hypergraph matchings. *In Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques* (pp. 10-20). Springer Berlin Heidelberg.

Auletta, V., De Prisco, R., Penna, P. and Persiano, G., 2004, March. Deterministic truthful approximation mechanisms for scheduling related machines. *In Annual Symposium on Theoretical Aspects of Computer Science* (pp. 608-619). Springer Berlin Heidelberg.

Azizoglu, M. and Kirca, O., 1998. Tardiness minimization on parallel machines. *International Journal of Production Economics*, 55(2), pp.163-168.

Baker, K.R. and Su, Z.S., 1974. Sequencing with duedates and early start times to minimize maximum tardiness. *Naval Research Logistics (NRL)*, 21(1), pp.171-176.

Balakrishnan, N., Kanet, J.J. and Sridharan, V., 1999. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers & Operations Research*, 26(2), pp.127-141.

Ball, M., Barnhart, C., Nemhauser, G. and Odoni, A., 2007. Air transportation: Irregular operations and control. *Handbooks in Operations Research and Management Science*, 14, pp.1-67.

Ball, M.O. and Lulli, G., 2004. Ground delay programs: Optimizing over the included flight set based on distance. *Air Traffic Control Quarterly*, 12(1), pp.1-25.

Bansal, N. and Sviridenko, M., 2006, May. The santa claus problem. *In Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing* (pp. 31-40). ACM.

Barnhart, C., Bertsimas, D., Caramanis, C. and Fearing, D., 2012. Equitable and efficient coordination in traffic flow management. *Transportation Science*, 46(2), pp.262-280.

Bertsimas, D., Farias, V.F. and Trichakis, N., 2011. The price of fairness. *Operations Research*, 59(1), pp.17-31.

Bertsimas, D. and Patterson, S.S., 2000. The traffic flow management rerouting problem in air traffic control: A dynamic network flow approach. *Transportation Science*, 34(3), pp.239-255.

Bezáková, I. and Dani, V., 2005. Allocating indivisible goods. *ACM SIGecom Exchanges*, 5(3), pp.11-18.

Bisias, D., Lo, A.W. and Watkins, J.F., 2012. Estimating the NIH efficient frontier. *PloS one*, 7(5), p.e34569.

Biskup, D., Herrmann, J. and Gupta, J.N., 2008. Scheduling identical parallel machines to minimize total tardiness. *International Journal of Production Economics*, 115(1), pp.134-142.

Boche, H., Wiczanowski, M. and Stanczak, S., 2007. Unifying view on min-max fairness, max-min fairness, and utility optimization in cellular networks. *EURASIP Journal on Wireless Communications and Networking*, 2007(1), p.034869.

Bonald, T. and Massoulié, L., 2001, June. Impact of fairness on Internet performance. *In ACM SIGMETRICS Performance Evaluation Review* (Vol. 29, No. 1, pp. 82-91). ACM.

Bonald, T., Massouli, L., Proutiere, A. and Virtamo, J., 2006. A queueing analysis of max-min fairness, proportional fairness and balanced fairness. *Queueing Systems*, 53(1), pp.65-84.

Butler, M. and Williams, H.P., 2002. Fairness versus efficiency in charging for the use of common facilities. *Journal of the Operational Research Society*, pp.1324-1329.

Charny, A., 1994. An algorithm for rate allocation in a packet-switching network with feedback. Master's thesis. *MIT, Cambridge, MA*.

Che, Y.K. and Gale, I., 1996. Expected revenue of all-pay auctions and first-price sealed-bid auctions with budget constraints. *Economics Letters*, 50(3), pp.373-379.

Chen, B., Potts, C.N. and Woeginger, G.J., 1998. A review of machine scheduling: Complexity, algorithms and approximability. *In Handbook of Combinatorial Optimization* (pp. 1493-1641). Springer US.

Christodoulou, G. and Kovács, A., 2013. A deterministic truthful PTAS

for scheduling related machines. *SIAM Journal on Computing*, 42(4), pp.1572-1595.

Christodoulou, G., Kovács, A. and van Stee, R., 2010, December. A truthful constant approximation for maximizing the minimum load on related machines. *In International Workshop on Internet and Network Economics* (pp. 182-193). Springer Berlin Heidelberg.

Clarke, E.H., 1971. Multipart pricing of public goods. *Public choice*, 11(1), pp.17-33.

de Véricourt, F. and Zhou, Y.P., 2005. A routing problem for call centers with customer callbacks after service failure. *Operations Research*, 53(6), pp.968-981.

Deuermeyer, B.L., Friesen, D.K. and Langston, M.A., 1982. Scheduling to maximize the minimum processor finish time in a multiprocessor system. SIAM Journal on Algebraic Discrete Methods, 3(2), pp.190-196.

Dhangwatnotai, P., Dobzinski, S., Dughmi, S. and Roughgarden, T., 2011. Truthful approximation schemes for single-parameter agents. *SIAM Journal on Computing*, 40(3), pp.915-933.

Emmons, H., 1969. One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17(4), pp.701-715.

Epstein, L., Levin, A. and van Stee, R., 2013, January. A unified approach to truthful scheduling on related machines. *In Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 1243-1252). Society for Industrial and Applied Mathematics.

Epstein, L. and van Stee, R., 2010. Maximizing the minimum load for selfish agents. *Theoretical Computer Science*, 411(1), pp.44-57.

Garey, M.R. and Johnson, D.S., 1979. Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences.

Gibbard, A., 1973. Manipulation of voting schemes: a general result. *Econometrica: Journal of the Econometric Society*, pp.587-601.

Graham, R.L., 1969. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2), pp.416-429.

Goel, A., Meyerson, A. and Plotkin, S., 2000, May. Combining fairness with throughput: Online routing with multiple objectives. *In Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing* (pp. 670-679). ACM.

Gonzalez, T., Ibarra, O.H. and Sahni, S., 1977. Bounds for LPT schedules on uniform processors. *SIAM Journal on Computing*, 6(1), pp.155-166.

Gui, H., Müller, R. and Vohra, R.V., 2004. *Dominant strategy mechanisms with multidimensional types* (No. 1392). Discussion paper//Center for Mathematical Studies in Economics and Management Science.

Hahne, E.L., 1991. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in Communications*, 9(7), pp.1024-1039.

Hall, N.G., Kubiak, W. and Sethi, S.P., 1991. Earliness – tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date. *Operations Research*, 39(5), pp.847-856.

Heydenreich, B., Müller, R. and Uetz, M., 2010. Mechanism design for decentralized online machine scheduling. *Operations Research*, 58(2), pp.445-457.

Huang, X.L. and Bensaou, B., 2001, October. On max-min fairness and scheduling in wireless ad-hoc networks: analytical framework and implementation. *In Proceedings of the 2nd ACM International Symposium on Mobile Ad-hoc Networking & Computing* (pp. 221-231). ACM.

Jackson, J.R., 1955. *Scheduling a production line to minimize maximum tardiness*. CALIFORNIA UNIV LOS ANGELES NUMERICAL ANALYSIS RESEARCH.

Kayvanfar, V., Komaki, G.M., Aalaei, A. and Zandieh, M., 2014. Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times. *Computers & Operations Research*, 41, pp.31-43.

Kim, D.W., Na, D.G. and Chen, F.F., 2003. Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer-Integrated Manufacturing*, 19(1), pp.173-181.

Kleinberg, J., Rabani, Y. and Tardos, É., 1999. Fairness in routing and load balancing. *In Foundations of Computer Science, 1999. 40th Annual Symposium on* (pp. 568-578). IEEE.

Klemperer, P., 1999. Auction theory: A guide to the literature. *Journal of Economic Surveys*, 13(3), pp.227-286.

Koulamas, C. and Kyparisis, G.J., 2000. Scheduling on uniform parallel machines to minimize maximum lateness. Operations Research Letters, 26(4), pp.175-179.

Koutsoupias, E. and Papadimitriou, C., 1999, March. Worst-case equilibria. *In Stacs* (Vol. 99, pp. 404-413).

Kovács, A., 2005, October. Fast monotone 3-approximation algorithm for scheduling related machines. *In European Symposium on Algorithms* (pp. 616-627). Springer Berlin Heidelberg.

Kumar, A. and Kleinberg, J., 2000. Fairness measures for resource allocation. *In Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on* (pp. 75-85). IEEE.

Lavi, R., Mu'Alem, A. and Nisan, N., 2003, October. Towards a characterization of truthful combinatorial auctions. *In Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on* (pp. 574-583). IEEE.

Lenstra, J.K., 1976. Sequencing by enumerative methods.

Lenstra, J.K., Kan, A.R. and Brucker, P., 1977. Complexity of machine scheduling problems. Annals of Discrete Mathematics, 1, pp.343-362.

Lin, W.Y., Lin, G.Y. and Wei, H.Y., 2010, May. Dynamic auction mechanism for cloud resource allocation. *In Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on* (pp. 591-592). IEEE.

Luh, H.P. and Viniotis, I., 2002. Threshold control policies for heterogeneous server systems. *Mathematical Methods of Operations Research*, 55(1), pp.121-142.

Luo, H., Lu, S., Bharghavan, V., Cheng, J. and Zhong, G., 2004. A packet scheduling approach to QoS support in multihop wireless networks. *Mobile Networks and Applications*, 9(3), pp.193-206.

Ma, Q. and Steenkiste, P., 1997, October. On path selection for traffic with bandwidth guarantees. *In Network Protocols, 1997. Proceedings., 1997 International Conference on* (pp. 191-202). IEEE.

Mas-Colell, A., Whinston, M.D. and Green, J.R., 1995. *Microeconomic theory* (Vol. 1). New York: Oxford University Press.

Massoulié, L., 2007. Structural properties of proportional fairness: Stability and insensitivity. *The Annals of Applied Probability*, pp.809-839.

Megiddo, N., 1974. Optimal flows in networks with multiple sources and sinks. *Mathematical Programming*, 7(1), pp.97-107.

Mo, J. and Walrand, J., 2000. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking (ToN)*, 8(5), pp.556-567.

Myerson, R.B., 1981. Optimal auction design. *Mathematics of Operations Research*, 6(1), pp.58-73.

Nandagopal, T., Kim, T.E., Gao, X. and Bharghavan, V., 2000, August.

Achieving MAC layer fairness in wireless packet networks. *In Proceedings of the 6th Annual International Conference on Mobile Computing and Networking* (pp. 87-98). ACM.

Nisan, N. and Ronen, A., 1999, May. Algorithmic mechanism design. *In Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing* (pp. 129-140). ACM.

Porter, R., 2004, May. Mechanism design for online real-time scheduling. *In Proceedings of the 5th ACM Conference on Electronic Commerce* (pp. 61-70). ACM.

Potts, C.N. and Van Wassenhove, L.N., 1985. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2), pp.363-377.

Ramakrishnan, K., Jain, D. and Chiu, D., 1987. *A selective binary feedback scheme for general topologies.* Technical Report DEC-TR-510, Digital Equipment Corporation.

Rawls, J., 2009. *A theory of justice.* Harvard University Press.

Roberts, L., 1994. Enhanced PRCA (proportional rate-control algorithm). *In ATM Forum.*

Rowe, C.J. and Broadie, S., 2002. *Nicomachean ethics.* Oxford University Press, USA.

Saks, M. and Yu, L., 2005, June. Weak monotonicity suffices for truthfulness on convex domains. *In Proceedings of the 6th ACM Conference on Electronic Commerce* (pp. 286-293). ACM.

Sarkar, S. and Tassiulas, L., 2000, March. Fair allocation of discrete bandwidth layers in multicast networks. *In INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (Vol. 3, pp. 1491-1500). IEEE.

Satterthwaite, M.A., 1975. Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2), pp.187-217.

Sen, A., 1973. *On economic inequality*. Oxford University Press.

Sridharan, A. and Krishnamachari, B., 2004. Max-min fair collision-free scheduling for wireless sensor networks. *In Performance, Computing, and Communications, 2004 IEEE International Conference on* (pp. 585-590). IEEE.

Srinivasan, R. and Somani, A.K., 2003. On achieving fairness and efficiency in high-speed shared medium access. *IEEE/ACM Transactions on Networking (TON)*, 11(1), pp.111-124.

Su, X. and Zenios, S.A., 2006. Recipient choice can address the efficiency-equity trade-off in kidney transplantation: A mechanism design model. *Management Science*, 52(11), pp.1647-1660.

Tang, A., Wang, J. and Low, S.H., 2004, March. Is fair allocation always inefficient. In INFOCOM 2004. *Twenty-Third Annual Joint Conference of the IEEE Computer and Communications Societies* (Vol. 1). IEEE.

Vickrey, W., 1961. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of Finance*, 16(1), pp.8-37.

Williams, A., 1985. Economics of coronary artery bypass grafting. *Br Med J (Clin Res Ed)*, 291(6491), pp.326-329.

Yang, B., Geunes, J. and O'Brien, W.J., 2004. A heuristic approach for minimizing weighted tardiness and overtime costs in single resource scheduling. *Computers & Operations Research*, 31(8), pp.1273-1301.

Young, H.P., 1995. *Equity: in theory and practice*. Princeton University Press.

Zhang, H., Jiang, H., Li, B., Liu, F., Vasilakos, A.V. and Liu, J., 2016. A framework for truthful online auctions in cloud computing with heterogeneous

user demands. *IEEE Transactions on Computers*, 65(3), pp.805-818.